



# UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

---

## FACULTAD DE CIENCIAS

Clasificación de Pigmentos en Patrimonio cultural, utilizando  
redes neuronales convolucionales unidimensionales

### T E S I S

QUE PARA OBTENER EL TÍTULO DE:

Matemático

P R E S E N T A :

Roberto Damián López

DIRECTOR DE TESIS:

Dr. Oscar G. de Lucio Morales



## Índice

|                                                              |    |
|--------------------------------------------------------------|----|
| Índice .....                                                 | ii |
| 1. Introducción .....                                        | 1  |
| 2. Desarrollo teórico.....                                   | 3  |
| 2.1. Historia. ....                                          | 3  |
| 2.2. Marco Teórico.....                                      | 4  |
| 2.2.1. El perceptrón como compuerta universal booleana. .... | 6  |
| 2.2.2. <b>MLP</b> como Aproximadores universales. ....       | 11 |
| 2.2.3. <b>MLP</b> como clasificadores. ....                  | 15 |
| 2.2.4 Redes Neuronales Convolucionales. ....                 | 17 |
| 3. Optimización.....                                         | 21 |
| 3.1. Descenso por Gradiente.....                             | 22 |
| 3.2. <b>ADAM</b> . ....                                      | 22 |
| 3.3 Función de pérdida.....                                  | 23 |
| 4. Arquitecturas de Redes Neuronales.....                    | 25 |
| 4.1. LeNet5 (1998).....                                      | 25 |
| 4.2. Alexnet (2012). ....                                    | 28 |
| 4.3. VGG16 ( 2014 ). ....                                    | 29 |
| 4.4. <b>GoogleNet</b> (2014).....                            | 31 |
| 4.5 <b>ResNet</b> (2015).....                                | 31 |
| 4.6. Otras arquitecturas.....                                | 31 |
| 4.7. Arquitectura elegida. ....                              | 32 |
| 5. Desarrollo Práctico. ....                                 | 33 |
| 5.1. Imágenes Hiperespectrales. ....                         | 33 |
| 5.2. Espectros para clasificación.....                       | 34 |
| 5.3. Procesamiento de Datos. ....                            | 37 |
| 5.4. Entrenamiento y Clasificación de la red neuronal.....   | 39 |
| 5.4.1. Primera red entrenada. ....                           | 40 |
| 5.4.2 Segunda red entrenada. ....                            | 43 |
| 5.4.3 Tercera red entrenada. ....                            | 44 |

|                                                  |    |
|--------------------------------------------------|----|
| 6. Conclusiones.....                             | 51 |
| Apéndices.....                                   | 52 |
| Apéndice 1. Infraestructura.....                 | 52 |
| A.1. Hardware.....                               | 52 |
| A.1.2 Software.....                              | 52 |
| A.1.2.1. CUDA.....                               | 52 |
| A.1.2.2. Python.....                             | 53 |
| A.1.2.3. Pythorch.....                           | 54 |
| A.1.2.4. Spectral Python.....                    | 56 |
| A.1.2.5. PyQT5.....                              | 56 |
| A.1.3 Preparación de ambiente de desarrollo..... | 57 |
| Apendice 2. K-medias (K-means) .....             | 58 |
| Bibliografía y referencias:.....                 | 59 |

## Agradecimientos Personales

## Agradecimientos Académicos

Este trabajo fue apoyado por los siguientes proyectos:

**Proyecto CONACYT CF 2019 No. 731762**, denominado ***“PUNTO DE QUIEBRE: Estudio de contextos pirotecnológicos y sus implicaciones arqueológicas, arqueomagnéticas y físicas en la innovación tecnológica de la sociedad Maya prehispánica”***.

**Proyecto PAPIIT IG100424**, denominado ***“Estudio de las propiedades físicas, químicas y mineralógicas de cerámicas, estucos y herramientas arqueológicas del área Maya”***

Agradezco el apoyo del *Laboratorio Nacional de Ciencias para la Investigación y Conservación del Patrimonio Cultural LANCIC* sede Instituto de Física, para realizar este trabajo de investigación, a través de los proyectos **CONACYT LN315853, LN299076, LN 314846, CB239609**.

Agradezco el apoyo recibido del **Instituto de Física**, a través del *Programa de Becas del Instituto de Física (PBIF24)*, etapa 1, de abril a junio de 2024.

## 1. Introducción

El *Laboratorio Nacional de Ciencias para la Investigación y Conservación del Patrimonio Cultural LANCIC*, tiene como una de sus misiones “*Desarrollar nuevas tecnologías y materiales para la investigación de bienes culturales e implementar protocolos y estrategias acordes a las características de los materiales que forman parte del patrimonio cultural.*” Es en este ámbito donde se desarrolla mi tesis.

Las técnicas de imágenes espectrales tienen un papel importante en las ciencias de la conservación, destacan el uso de mapeos generados por espectroscopía de fluorescencia de rayos X (XRF), y el uso de imágenes hiperespectrales de reflectancia en el intervalo visible (típicamente de 400 – 1000 nanómetros (nm) y en el infrarrojo (entre 1000 – 2000 nm). De estas técnicas se obtienen imágenes multi-dimensionales, donde las primeras dos dimensiones son espaciales y las siguientes, corresponden a un vector *n-dimensional* conocido como **espectro** y que contiene información física o química del objeto estudiado. Cada píxel en la imagen contiene entonces información de un espectro asociado con algún pigmento o material. Es sobre este conjunto de píxeles que se realiza el análisis de clasificación material.

Tendremos en cuenta que cada píxel no es necesariamente un espectro de un material puro, sino que casi siempre corresponde a una mezcla (puede ser de diferentes pigmentos o de pigmento y algún medio), y debido a esto es que usaremos redes neuronales, ya que estas han demostrado ser muy eficientes para la clasificación no lineal. Con esta herramienta podría determinarse el tipo de pigmento usado, a partir de un conjunto de entrenamiento y así indicar las zonas en donde dicho material está presente, en términos prácticos podría usarse para determinar si una pintura tiene el tipo de pigmento usado por el pintor o tiene uno diferente (que podría haber sido aplicado posteriormente) o incluso una mezcla.

Este trabajo se enfocará en clasificar regiones en objetos de estudio previamente estudiados, para catalogar los pigmentos empleados por el artista o aplicados en campañas de restauración posteriores, y proveer de herramientas que sirvan para la clasificación de otras obras de arte semejantes (desde el punto de vista del conjunto de entrenamiento). Para esto, desarrollaremos y

utilizaremos un **clasificador con una red neuronal convolucional unidimensional (1D-CNN)**. Explicaremos el fundamento teórico de su uso y las ventajas de éste. Como parte complementaria se desarrolla en lenguaje PYTHON, un software con una interfaz gráfica para que, al utilizar la red neuronal, sea posible realizar estas clasificaciones.

## 2. Desarrollo teórico

### 2.1. Historia.

El termino **Inteligencia Artificial (IA)** tiene sus inicios en el trabajo de A. M. Turing (1950)<sup>[21]</sup>: “COMPUTING MACHINERY AND INTELLIGENCE” en el que, en los primeros renglones nos pregunta “¿Pueden las máquinas pensar?” y partiendo de esta idea ofrece un método para determinar si es posible que una máquina piense, este método es la conocida “Prueba de Turing”. Desde entonces cuando se puede hacer que una computadora sea capaz de realizar alguna tarea humana y que estas tareas puedan mejorarse conforme recopilan información, le llamamos inteligencia artificial. Una de las ramas de la **IA** es el Aprendizaje de Máquina o **Machine Learning (ML)** cuyo término fue acuñado en 1959 por Arthur Samuel:

*“Entonces, ¿por qué el nombre de "Machine Learning" (aprendizaje de máquinas)? La razón, como sucede en muchas ocasiones, es el marketing: Arthur Samuel, un pionero en el campo de juegos de computadora y la inteligencia artificial, acuñó el término en 1959 mientras era parte de IBM. De manera similar como cuando IBM en la década de 2010s intentó introducir el término "computación cognitiva" para diferenciarse de la competencia, en los años 1960s, IBM usó el nuevo término llamativo Machine Learning para atraer tanto a clientes como a empleados talentosos.”* <sup>[29]</sup>

Aprendizaje de Máquina incluye muchas técnicas de estadística inferencial, una de estas es la red neuronal cuyo desarrollo abarca desde mediados del siglo XX.

En 1943 Warren McCulloch y Walter Pitts, escribieron un artículo en el que proponen un modelo de cómo el cerebro humano podría funcionar, en este **trabajo** concluyen que “En su propio ámbito, la diferencia entre redes equivalentes y redes equivalentes en sentido estricto, indica el uso apropiado y la importancia de estudios temporales sobre la actividad nerviosa: a los que la teoría biofísica-matemática aporta una herramienta para el tratamiento simbólico de las redes estudiadas y un método fácil para construir redes hipotéticas de propiedades específicas.” <sup>[16]</sup>. Este

trabajo es tomado por Frank Rosenblatt quien en 1958 escribió el artículo *"The perceptron: a probabilistic model for information storage and organization in the brain"* <sup>[21]</sup>, y creó un programa para implementarlo. El mismo año Frank Rosenblatt escribió en el periódico The New York Times: *"este es (Perceptron) el embrión de un ordenador electrónico que espera poder caminar, hablar, ver, escribir, reproducirse y tener consciencia de su existencia"* <sup>[8]</sup>. En 1959 Bernard Widrow y Marcian Hoff de Stanford **desarrollaron** 2 modelos llamados **"ADALINE"** y **"MADALINE"** que fueron de los primeros modelos utilizados en aplicaciones de la vida real. En el transcurso del tiempo se desarrollaron diferentes métodos de aprendizaje para las redes neuronales y fue así que, en la década de los 80's aparecen las primeras redes neuronales multicapa. Es en la década del 2010 que el auge de arquitecturas más funcionales y poderosas llegan a la escena: **AlexNet** (2012), **GoogleNet** (2014), **ResNet** (2015), **Inception** (2015), **VGG16** (2016). En la actualidad hay muchos trabajos sobre modelos y sus usos, **por lo que** tomaremos como base el desarrollo que se ha hecho para el procesamiento de imágenes hiperespectrales con redes neuronales convolucionales de una dimensión, para eso explicaremos el funcionamiento general de estas.

## 2.2. Marco Teórico.

La neurona abstracta (Figura 1) es la base de cualquier red neuronal, esta unidad imita el funcionamiento de una neurona biológica, que consta de una señal de entrada, pesos sinápticos y una función de activación. Definiremos entonces una neurona abstracta para formalizar el objeto que vamos a utilizar.

**Definición 1:** Una neurona abstracta es una tupla  $(\mathbf{x}, \mathbf{w}, \phi, y)$ , donde  $\mathbf{x}^T = (x_0, x_1, \dots, x_n)$  es el vector de entrada,  $\mathbf{w}^T = (w_0, w_1, \dots, w_n)$  es el vector de pesos sinápticos, con  $x_0 = -1$  y  $w_0 = b$ , el sesgo, y  $\phi$  es una función de activación que define la función de resultado  $y = \phi(\mathbf{x}^T \mathbf{w}) = \phi(\sum_{i=0}^n w_i x_i)$ .

La salida  $y$  es comparada con un vector  $z$  de la misma dimensión que tiene los valores de las clases base. Esta comparación es un proceso que se realiza varias veces, a este proceso se le llama "aprendizaje". Dentro del aprendizaje de maquina hay dos tipos de aprendizaje: "Aprendizaje Supervisado" y "Aprendizaje No Supervisado". Las redes neuronales utilizan el "Aprendizaje Supervisado", que consiste en dar un valor inicial aleatorio al vector de pesos, pasar un valor de entrada  $x$  controlado, comparar con  $z$  valor perteneciente al conjunto de valores esperados en la

clasificación y modificar el vector  $w$  de pesos para que el error sea muy pequeño. Este tema lo abordaremos con detenimiento más adelante.

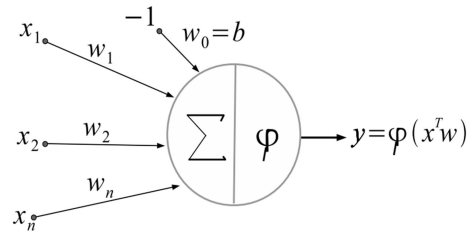


Figura 1. Representación simple de una neurona abstracta. Observamos un  $x$  un vector valores iniciales, el vector  $w$  de pesos iniciales, la neurona que realiza la consolidación de los valores y la función de activación que determina como ha de responder la neurona.

El perceptrón es una neurona abstracta obtenida por una implementación específica de la función de activación  $\phi$ ; la función de Heaviside, que se muestra a continuación:

$$\phi(X) = \begin{cases} 0, & \text{si } X < 0 \\ 1, & \text{si } X \geq 0 \end{cases} \quad (1)$$

Así, la salida del modelo del perceptrón se puede modelar:

$$y = \phi(x^T w - \theta) = \begin{cases} 1, & \text{si } \left( \sum_{i=0}^n w_i * x_i \right) - \theta < 0 \\ 0, & \text{si } \left( \sum_{i=0}^n w_i * x_i \right) - \theta \geq 0 \end{cases} \quad (2)$$

El uso más simple del perceptrón es en la implementación de las compuertas lógicas **AND** y **NOT**, que son las básicas para describir cualquier función booleana, este hecho lo discutiremos un poco más adelante.

### 2.2.1. El perceptrón como compuerta universal booleana.

Empecemos con una definición.

**Definición 2:** El espacio de aridad  $n$  de funciones booleanas es el conjunto que contiene todas las funciones booleanas posibles de aridad  $n$ . Una función de aridad  $n$  es un mapeo de la forma  $\{0,1\}^n \rightarrow \{0,1\}$ . Donde  $\{0, 1\}$  es el conjunto Booleano  $B$  y  $n$  es un numero natural.

Con esta definición tenemos que para el conjunto de funciones de aridad 2, el número de funciones está dado por  $2^{2^2} = 16$ . Estas funciones son descritas en la tabla 1:

| $(x, y)$ | $f_0$ | $f_1$ | $f_2$ | $f_3$ | $f_4$ | $f_5$ | $f_6$ | $f_7$ | $f_8$ | $f_9$ | $f_{10}$ | $f_{11}$ | $f_{12}$ | $f_{13}$ | $f_{14}$ | $f_{15}$ |
|----------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|----------|----------|----------|----------|----------|----------|
| (0, 0)   | 0     | 0     | 0     | 0     | 0     | 0     | 0     | 0     | 1     | 1     | 1        | 1        | 1        | 1        | 1        | 1        |
| (0, 1)   | 0     | 0     | 0     | 0     | 1     | 1     | 1     | 1     | 0     | 0     | 0        | 0        | 1        | 1        | 1        | 1        |
| (1, 0)   | 0     | 0     | 1     | 1     | 0     | 0     | 1     | 1     | 0     | 0     | 1        | 1        | 0        | 0        | 1        | 1        |
| (1, 1)   | 0     | 1     | 0     | 1     | 0     | 1     | 0     | 1     | 0     | 1     | 0        | 1        | 0        | 1        | 0        | 1        |

Tabla 1. Tabla de funciones booleanas de aridad 2.

En la tabla claramente vemos la función Falso ( $f_0$ ) y la función Verdad ( $f_{15}$ ) así como la función conjunción ( $f_1$ ), estas funciones son fácilmente implementadas por un perceptrón, por ejemplo, la función conjunción sería implementada si la función de transferencia es:

$$\varphi((X, Y) \cdot (1, 1) - 2) = \begin{cases} 0, & \text{si } (X, Y) \cdot (1, 1) - 2 < 0 \\ 1, & \text{si } (X, Y) \cdot (1, 1) - 2 \geq 0 \end{cases} \quad (3)$$

Donde  $x^T = (X, Y)$ ,  $w = (1, 1)$  y  $\theta = 2$ . Así el perceptrón equivale a la compuerta lógica de la conjunción, y de igual manera podemos hacer con las demás funciones lógicas, adecuando la función. Entonces podríamos ya tener las funciones básicas que pueden expresar las demás funciones booleanas

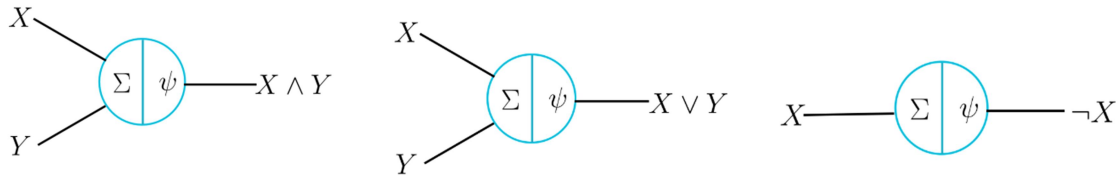


Figura 2. Compuertas lógicas representadas por perceptrones. De izquierda a derecha; compuerta AND, OR y Negación.

En el caso de la compuerta conjunción, si la representamos en un plano cartesiano con los puntos que son falso como azules y los verdaderos como rojo y graficamos la recta que representa a la neurona veremos que la función de activación define 2 regiones: una superior que incluye al punto rojo y una región azul que serían los puntos azules. A este tipo de funciones comúnmente se le llama funciones booleanas linealmente separables, pues una recta divide en el plano a los puntos de acuerdo con la clasificación verdadero o falso. En la Figura 3. tenemos esta representación.

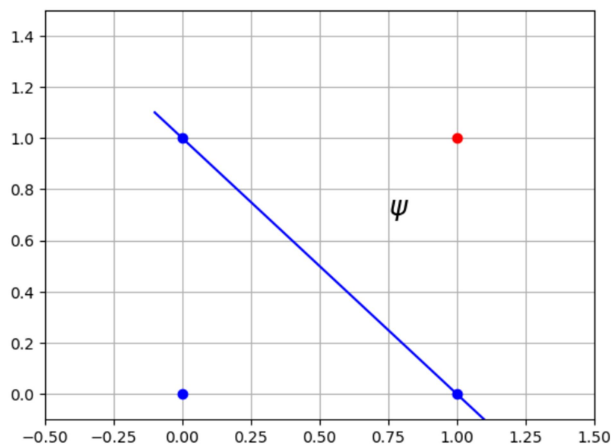


Figura 3. Representación de una función linealmente separable. Grafica realizada con Python.

En el caso de las funciones booleanas de aridad 2, de las 16 funciones hay 2 que no son linealmente separables, la función  $f_6(\neg\text{XOR})$  y la  $f_9(\text{XOR})$ . Sin embargo, hay forma de representarla con neuronas abstractas, y sería de la siguiente manera:

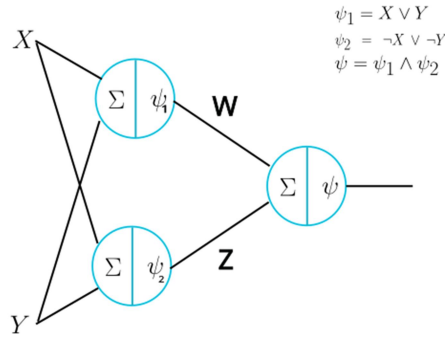


Figura 4. Representación de la función booleana  $f_9$  de aridad 2.

La única característica que cambió fue que, para representar esta función booleana, se necesitó hacer que la salida de dos funciones booleanas fueran la entrada de una tercera. A esta configuración se la llama *Perceptrón Multicapa (MLP)* y se llama multicapa porque entre la capa de entrada y la de salida hay una capa de neuronas abstractas antes de la neurona del resultado. Con la inclusión de esta configuración podríamos decir que la expresividad de una red neuronal ha aumentado, y podríamos describir funciones booleanas más complejas, por ejemplo:

Dada  $(X_1, X_2, X_3, X_4, X_5)$  describamos la función:

$$((X_4 \wedge \neg X_1 \wedge X_3) \vee (X_4 \wedge \neg X_2)) \wedge ((X_1 \wedge X_2) \vee \neg(X_1 \wedge X_3)) \quad (4)$$

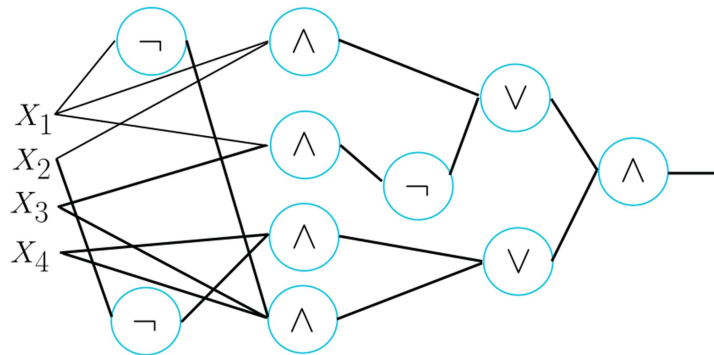


Figura 5. Representación de una función booleana más compleja.

Como vemos, con un **MLP** pudimos expresar la función booleana propuesta. Sin embargo, podríamos eliminar las capas que hacen la negación, así si cambiamos los parámetros de peso en

$w = (1, -1)$  o  $w = (-1, 1)$  según el caso, el resultado sería como el mostrado en la figura 6, llamamos a  $\Lambda_1$  como la neurona que ha cambiado sus pesos mencionados anteriormente y la neurona  $V_1$  de igual manera tendría un valor de pesos modificado para simplificar:

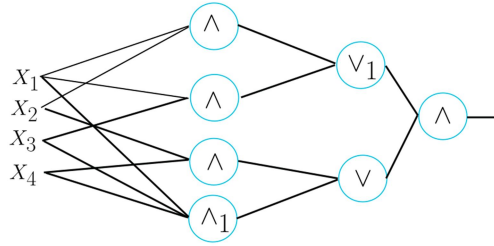


Figura 6. Simplificación de la función booleana.

Con esto nos damos cuenta que hay varias formas de representarla, incluso si usamos la forma normal disyuntiva, tendríamos:  $(\neg X_1 \wedge X_3 \wedge X_4) \vee (X_2 \wedge X_4)$ . Esta sería una red más simple.

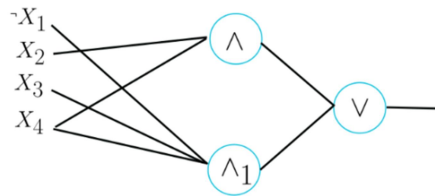


Figura 7. Forma normal disyuntiva de la función booleana.

Observemos que al tener menos neuronas la complejidad de su implementación y los recursos utilizados podrían ser menos, pero no siempre es este el caso. A la fórmula booleana de aridad 2  $f_9$ , se le conoce como la función XOR o conjunción exclusiva y es denotada por el símbolo  $\oplus$ , entonces la siguiente fórmula :

$$X_1 \oplus X_2 \oplus X_3, \quad (5)$$

cuya representación en **MLP** sería:

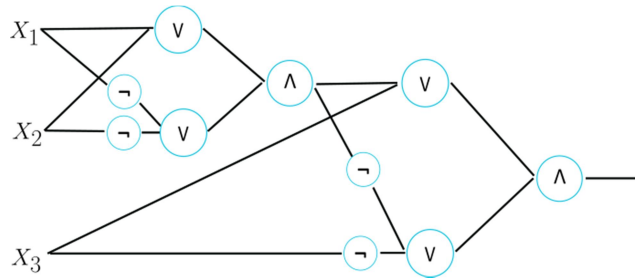


Figura 8. Representación de la ecuación 5 en MLP.

En el álgebra booleana hay formas de estandarizar las funciones de tal manera que al escribir una formula esta tenga una representación que tenga la menor redundancia posible, de estas formas elegiremos la Forma Normal Disyuntiva (**FND**) para simplificarla y lo que nos resulta es:

$$(X_1 \wedge X_2 \wedge X_3) \vee (X_1 \wedge \neg X_2 \wedge \neg X_3) \vee (\neg X_1 \wedge X_2 \wedge \neg X_3) \vee (\neg X_1 \wedge \neg X_2 \wedge X_3) \quad (6)$$

Y su representación en **MLP** es:

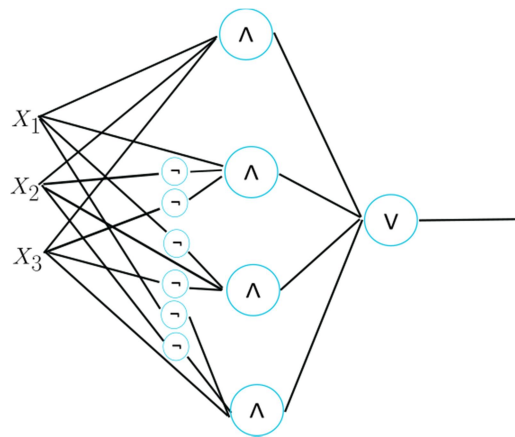


Figura 9. Representación de FND de la formula 5 en MLP.

Como podemos ver, aunque el número de capas ocultas disminuyó, el número de neuronas aumentó, esto en cuestión de recurso y complejidad computacional aumentan. Por lo que la profundidad de la red depende de la complejidad que arroje la fórmula, y no hay una regla para poder simplificar la fórmula que resulte en una mejora computacional. Por esta razón, obtener un algoritmo que pueda encontrar una configuración de **MLP** que computacionalmente sea más eficiente depende de la función a ser modelada.

Las MLP pueden modelar cualquier función booleana, pero hay que tomar en cuenta la función para obtener una representación adecuada de la función booleana. Este es aun uno de los problemas que quedan abiertos, es decir, ¿Habr  una forma de establecer la arquitectura correcta para una funci n booleana cualquiera?

### 2.2.2. *MLP* como Aproximadores universales.

En el a o de 1900 David Hilbert en su problema n mero 13 establece: “*la imposibilidad de resolver la ecuaci n general de grado 7 en t rminos de funciones de dos argumentos*”. Pasaron 57 a os hasta que Kolmogorov demostr  que Hilbert estaba equivocado, en una serie de resultados que culminaron con el Teorema de Superposici n de Kolmogorov, que nos permite escribir una funci n multivariada como suma y composici n de funciones monovariadas. Cabe mencionar que Kolmogorov en su demostraci n no da un m todo constructivo para dicha funci n, solo propone su existencia.

Para hacerlo son varios los caminos que tomaron diferentes personajes. Sin embargo, son George Cybenko y Kurt Hornik quienes proponen un m todo que se puede aplicar perfectamente a las redes neuronales para un uso m s amplio.

G. Cybenko en su art culo de 1989 “*Approximation by Superposition of a Sigmoidal Function*” [5] propone que cualquier funci n continua de  $n$  variables con soporte en el hipercubo unitario puede ser aproximada por una combinaci n lineal finita de composiciones de una funci n de una sola variable y un conjunto de funcionales afines.

G. Cybenko hace las siguientes definiciones usando la siguiente notaci n: Sea  $I_n$  el cubo  $n$  dimensional  $[0, 1]^n$ ,  $C(I_n)$  el espacio de funciones continuas en  $I_n$  y  $M(I_n)$  el espacio de medidas de Borel con signo sobre  $I_n$ , entonces tenemos:

**Definici n 3:** Decimos que  $\sigma$  es discriminatoria si para una medida  $\mu \in M(I_n)$

$$\int_{I_n} \sigma(y^T x + \theta) d\mu(x) = 0 \quad (7)$$

Para toda  $y \in \mathbb{R}^n$  y  $\theta \in \mathbb{R}$  implica que  $\mu = 0$ .

**Definición 4:** Decimos que  $\sigma$  es sigmoidal si

$$\sigma(x) \begin{cases} 1 & \text{si } x \rightarrow +\infty \\ 0 & \text{si } x \rightarrow -\infty \end{cases} \quad (8)$$

Su principal resultado es el siguiente teorema:

**Teorema 1 (Cybenko):** Sea  $\sigma$  una función sigmoidal continua, entonces cualquier suma finita de la forma

$$G(x) = \sum_{i=1}^n \alpha_i \sigma(y_i^T x + \theta_i) \quad (9)$$

Es densa en  $C(I_n)$ . Es decir, dada una función  $f \in C(I_n)$  y  $\varepsilon > 0$ , hay una suma,  $G(x)$  de la forma anterior tal que

$$|G(x) - f(x)| < \varepsilon, \forall x \in I_n \quad (10)$$

Para demostrar este teorema, Cybenko primero demuestra que si la función  $\sigma$  es discriminatoria en la suma finita de la forma (7) entonces es densa y que las funciones de tipo sigmoidal son discriminatorias. La demostración en el artículo es muy clara y no la reproduciremos aquí<sup>[5]</sup>.

Dado que la suma finita de la forma (9) es prácticamente la definición de una red neuronal de una sola capa (*comparando con la ecuación (2) que es una neurona, así la suma de neuronas actúa como la suma finita de neuronas, es decir, es una red neuronal de una sola capa*). Hay que hacer una precisión; aunque una red neuronal puede aproximar cualquier función arbitraria de complejidad no especificada, se requiere que el número de neuronas y sus parámetros sean ajustados dependiendo de la función que se desea aproximar.

En favor de la intuición hagamos un pequeño ejercicio que muestre como con funciones sigmoidales se puede representar una función:

Sea  $f(x) = \frac{(3x-1)^3 - 3x+3}{4}$  definida sobre el intervalo  $[0, 1]$  cuya grafica se muestra en la figura:

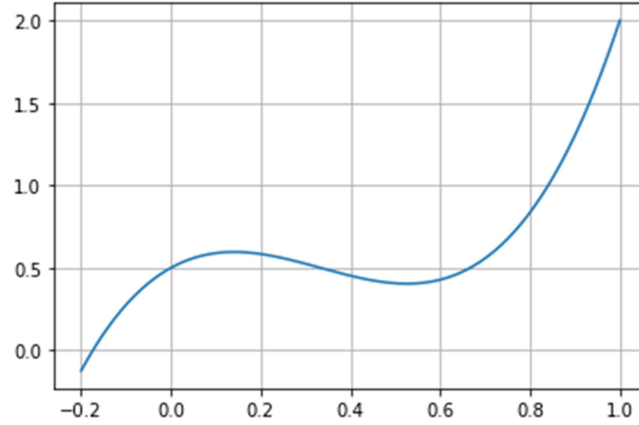


Figura 10. Función del  $f(x)$  del ejemplo.

Vamos a aproximar esta función con una función como la que se especifica en el teorema, a la combinación lineal de funciones, como se piden, las asociaremos de manera natural con la representación de un MLP, ya que la formulación de esta es exactamente la misma, así intercambiaremos en el ejemplo MLP, red neuronal o combinación lineal de funciones de una variable. Primero veremos que esta función esté acotada en el intervalo  $[0, 1]$ , observamos cómo afectan los parámetros si usamos una red neuronal con dos neuronas. Nuestra red neuronal está dada por

$$G(x) = \sum_{i=1}^n \alpha_i \sigma(w_i x + b_i). \quad (11)$$

donde  $n$  es el número de nodos,  $w$  es el vector con los valores de los pesos de entrada a el nodo y,  $\alpha$  corresponde a los pesos de salida del nodo hacia el resultado y es el parámetro que como veremos regulará la altura de la función,  $w$  y  $b$  son los que modelan la función sigmoide.

Para esta red de dos nodos es decir  $n = 2$ , es fácil ajustar los valores de los parámetros pues sabemos los valores que la función debe de tener y también sabemos el valor de los intervalos. Con  $\alpha = (f(\frac{1}{2}), f(\frac{1}{2}))$  es decir a la mitad del intervalo,  $w = (600, 600)$  elegidos de manera intuitiva y  $b = (0, -1)$  los valores que son los términos independientes. Utilizaremos como sigma la función sigmoideal

$$\sigma(x) = \frac{1}{1 - e^{-x}} \quad (12)$$

Si sustituimos en la ecuación 11 los valores  $\alpha$ ,  $w$ ,  $b$  para  $n = 2$  y hacemos los cálculos tenemos la siguiente gráfica de  $G(x)$ , comparada con  $f(x)$ :

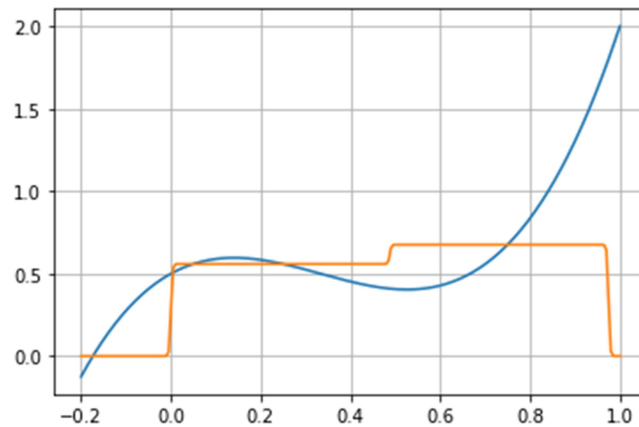


Figura 11. Función  $f(x)$  aproximada por  $G(x)$  para  $n = 2$ .

Para la siguiente iteración haremos  $n = 4$ , es decir el número de nodos se duplica, utilizando los mismos parámetros del aumento de nodos tendríamos:

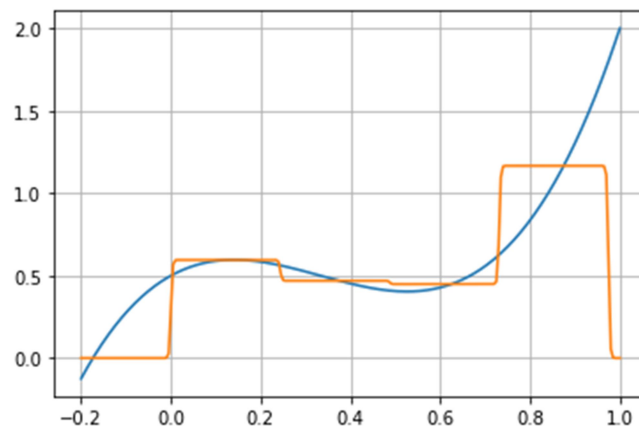


Figura 12. Iteración con  $n=4$ .

Si aumentamos el número de nodos de una manera más rápida veríamos que para 32 nodos la aproximación de la función  $G(x)$  sería así:

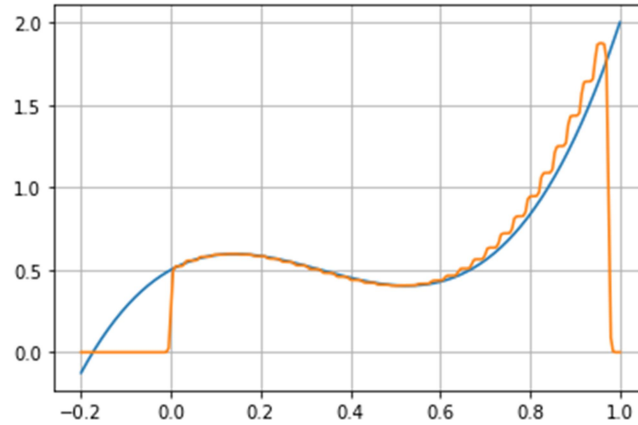


Figura 13. Iteración con  $n=32$ .

Así pues tenemos que si queremos una **mejor aproximación**, tendríamos que aumentar el número de nodos. En las conclusiones de Kurt Hornik<sup>[9]</sup>, que implementa una demostración para una red neuronal multicapa utilizando el Teorema de Stone-Weierstrass y la función cosenoidal aplanadora de Gallant y White, se establece que una red neuronal multicapa secuencial es capaz de aproximar cualquier función medible a un grado de aproximación arbitrario: *“Hemos establecido que tales redes ‘mapeadoras’ son aproximadores universales. Esto implica que cualquier falta de éxito en aplicaciones, se deberá a la falta de un aprendizaje adecuado, un número insuficiente de capas ocultas o la falta de una relación determinística entre la entrada y la salida”*<sup>[9]</sup>.

### 2.2.3. *MLP* como clasificadores.

En el área de Aprendizaje de Máquina, la clasificación es el proceso de mapear una función cuya entrada es una muestra y la salida es una etiqueta, hagamos pues una definición más formal:

**Definición 5: Población.** Es un conjunto cuyos elementos poseen características de interés para su estudio.

**Definición 6: Muestra.** Es un subconjunto de una población.

**Definición 7: Clasificación.** Es el proceso de encontrar un mapeo  $f$  cuyo dominio es una población  $X$  y su Imagen es un conjunto de características (clases)  $Y$ .

El mapeo puede no ser una función, pues hay ocasiones que, en la clasificación de un elemento de la población, se requiere que tenga 2 o más etiquetas(clases). A este tipo de clasificación se le llama clasificación *multi-etiqueta*. Para que las **MLP** puedan servir como clasificadores es necesario que utilicemos su propiedad de aproximadores universales, es decir, que tal mapeo sea una función, cuya imagen es un conjunto de dos o más clases. En el caso de dos clases se conoce como *clasificación binaria*, en el caso de más de dos clases se llama clasificación *multi-clase*(Tabla 2).

| Clasificación  | Población | clases                         | instancia          |
|----------------|-----------|--------------------------------|--------------------|
| Multi-etiqueta | $x \in X$ | $Y = \{(a, b, c), (a, c, d)\}$ | $f(x) = (a, c, d)$ |
| Multi-clase    | $x \in X$ | $Y = \{a, b, c, d, e\}$        | $f(x) = c$         |
| Binaria        | $x \in X$ | $Y = \{0, 1\}$                 | $f(x) = 1$         |

Tabla 2

El problema que abordaremos es el de aproximar una función dando los valores especificados por una muestra de nuestra población a clasificar, usando técnicas de regresión, optimización y aprendizaje de máquina. Para esto tomaremos a las poblaciones que pueden ser modeladas de forma cuantitativa, así cada elemento de dicha población puede ser visto como un vector n-dimensional en donde cada entrada del vector es una característica que se ha podido cuantificar.

## 2.2.4 Redes Neuronales Convolucionales.

Las Redes Neuronales Convolucionales (Convolutional Neural Network-CNN), son utilizadas principalmente en problemas de visión artificial, específicamente en el reconocimiento de objetos. Empecemos con una revisión sobre el uso de la convolución, como es adaptada para el procesamiento de señales y posteriormente como es utilizada en la arquitectura de un MLP para generar una red neural Convolucional.

La operación de convolución está definida tanto para sistemas discretos como para sistemas continuos. Nosotros utilizaremos la forma discreta en su aplicación de la siguiente manera:

$$f(x) * h(x) = \sum_{i=0}^n f(i)h(x - i) \quad (13)$$

La forma de cómo funciona este operador no la vamos a desarrollar pues hay bastante literatura y videos en internet. En cambio nos fijaremos como es utilizada en el procesamiento de señales donde es muy utilizada para hacer filtros, o funciones que modifican el comportamiento de una señal para resaltar, suavizar o generar una tercera señal. A continuación vemos un ejemplo de aplicación:

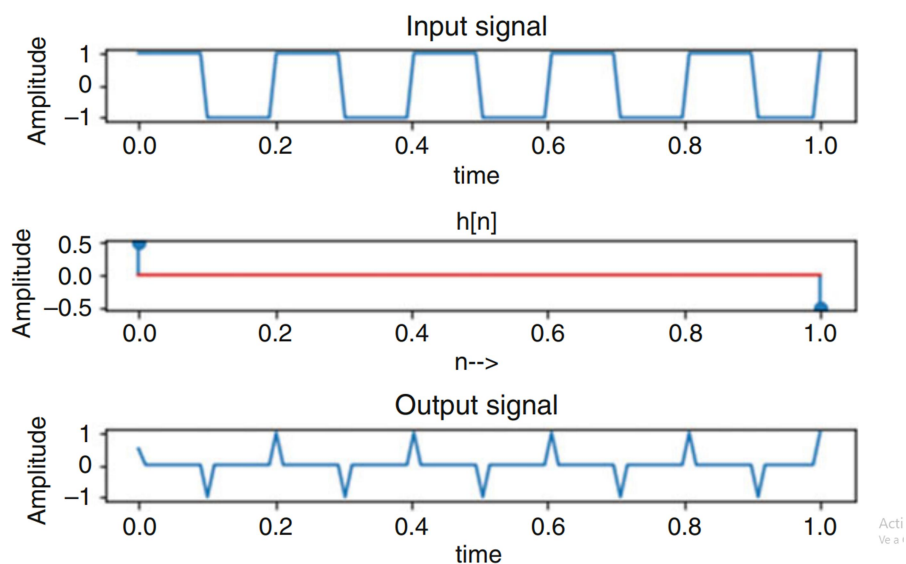


Figura 14. Ejemplo de aplicación de un filtro de paso alto a una señal cuadrada. El filtro de paso alto es una función  $h(n) = \{1/2, -1/2\}$ . Esta operación es una convolución. Imagen del Libro "Digital Signal Processing"<sup>[7]</sup> de Esakkirajan et al.

Uno de sus usos es la reducción de dimensión en imágenes y funciona así; Dada una imagen X (en el ejemplo tomaremos una de 16 x 16), usaremos un kernel (filtro) H de 4x4 para reducir su dimensión y conservar sus características importantes de la siguiente manera.

Haremos que el primer renglón sea el superior y empezamos a contar de izquierda a derecha, entonces la primera fila estaría con la nomenclatura  $x_{11}, x_{12}, \dots, x_{116}$ , para el primer renglón. Para el segundo:  $x_{21}, x_{22}, \dots, x_{216}$ , y así con los siguientes. Por último el kernel tendrá la misma nomenclatura;  $h_{11}, h_{12}, h_{21}, h_{22}$ .

La salida Y, se calcula tomando bloques del tamaño del kernel H en X y aplicándole el producto interno canónico por H, y desplazándose de tal manera que el kernel no ocupe una posición ya calculada anteriormente, a este desplazamiento se le llama **paso s**, en nuestro ejemplo el **paso** tiene el valor  $s = 2$ , así los componentes de la matriz vistos como bloque son:

$$X = \begin{pmatrix} \begin{pmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \end{pmatrix} & \dots & \begin{pmatrix} x_{115} & x_{116} \\ x_{215} & x_{216} \end{pmatrix} \\ \vdots & \ddots & \vdots \\ \begin{pmatrix} x_{151} & x_{152} \\ x_{161} & x_{162} \end{pmatrix} & \dots & \begin{pmatrix} x_{1515} & x_{1516} \\ x_{1615} & x_{1616} \end{pmatrix} \end{pmatrix} \quad (14)$$

Y la aplicación del filtro sería:

$$Y = \begin{pmatrix} \begin{pmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \end{pmatrix} \begin{pmatrix} h_{11} & h_{12} \\ h_{21} & h_{22} \end{pmatrix} & \dots & \begin{pmatrix} x_{115} & x_{116} \\ x_{215} & x_{216} \end{pmatrix} \begin{pmatrix} h_{11} & h_{12} \\ h_{21} & h_{22} \end{pmatrix} \\ \vdots & \ddots & \vdots \\ \begin{pmatrix} x_{151} & x_{152} \\ x_{161} & x_{162} \end{pmatrix} \begin{pmatrix} h_{11} & h_{12} \\ h_{21} & h_{22} \end{pmatrix} & \dots & \begin{pmatrix} x_{1515} & x_{1516} \\ x_{1615} & x_{1616} \end{pmatrix} \begin{pmatrix} h_{11} & h_{12} \\ h_{21} & h_{22} \end{pmatrix} \end{pmatrix} \quad (15)$$

Cada  $y_{ij}$  está afectado por el paso s y se podría determinar así:

$$y_{ij} = \begin{pmatrix} x_{si-1,sj-1} & x_{si-1,sj} \\ x_{si,sj-1} & x_{si,sj} \end{pmatrix} \cdot \begin{pmatrix} h_{11} & h_{12} \\ h_{21} & h_{22} \end{pmatrix} \quad (16)$$

Que al realizar el producto interno quedaría

$$y_{ij} == x_{si-1,sj-1} \cdot h_{11} + x_{si-1,sj} \cdot h_{12} + x_{si,sj-1} \cdot h_{21} + x_{si,sj} \cdot h_{22} \quad (17)$$

y el resultado se ve en la figura 15:

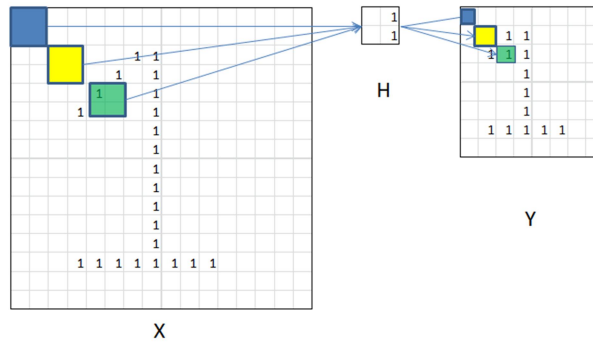


Figura 15. Ejemplo de reducción de dimensión. Al aplicar el filtro  $H(4 \times 4)$  a la entrada  $X(16 \times 16)$  el resultado es de menor dimensión ( $8 \times 8$ ), pero conserva algunas características del original.

Esta operación, variando más parámetros es lo que vamos a llamar convolución. Aunque por ejemplo en el sitio de la documentación de PYTORCH, establecen que la función `Conv1D` aplica una convolución a los vectores suministrados aplicando una operación de correlación cruzada, que es parecida pero en términos estrictos no es una convolución. Aun así el término es ampliamente usado y la convolución es utilizada para resaltar algunos atributos en imágenes y señales. Como ejemplo están los siguientes:



Imagen original



Imagen procesada

Figura 16. Ejemplo de imagen procesada con funciones de python para aplicar la operación convolución. Imagen obtenida del libro “Hybrid Imaging and Visualization” de Joseph Awange , Béla Paláncz , Lajos Völgyesi. Springer

La arquitectura de este tipo de redes es muy simple; básicamente se dividen en una **capa de entrada**, que contiene los datos a clasificar, una **capa convolucional** que es donde se implementa el filtro o kernel a ser usado, el resultado de esta capa siempre es denominado “feature map” o “mapeo de características”, una **función de activación** generalmente alguna función que disminuya la linealidad y provea gradientes no saturados con valores positivos, una **capa de agrupación** a la que es más frecuente denominar “pooling layer” que es una capa donde se comprime o reduce la amplitud de la capa de activación, una **capa de componentes totalmente conectados** que es

básicamente donde se realiza la clasificación y **el resultado** es una capa de salida, con datos que determinan la clasificación. Más adelante veremos más de este tema.

### 3. Optimización.

La optimización es minimizar o maximizar una función  $f(x)$  alterando los valores de  $x$ . A la función  $f$  la llamaremos función Objetivo. Hay dos consideraciones que se tendrán, la primera es tratar a la función objetivo como un oráculo durante el proceso de minimización, de esta forma tendremos acceso a la función sin necesidad de definirla, y, como sabemos los valores esperados, podremos ajustar una y otra vez hasta que encontremos los valores que necesitamos, de tal manera que definiremos los tipos de oráculos usados para tal aproximación.

**Definición 8:** Oráculo de valor: Oráculo de valor, oráculo de evaluación u oráculo de 0-grados para una función  $f: R^n \rightarrow R$  es un procedimiento tal que al ser consultado en un punto  $x \in R^n$  conocemos el valor de  $f$ .

**Definición 9:** Oráculo de gradiente: Oráculo de gradiente para una función  $f: R^n \rightarrow R$  es un procedimiento tal que al ser consultado en un punto  $x \in R^n$  conocemos el gradiente  $\nabla f(x) \in R^n$ , donde  $[\nabla f(x)]_i = \frac{\partial}{\partial x_i} f(x)$ .

**Definición 10.** Oráculo de primer orden: Oráculo 1er-grado para una función  $f: R^n \rightarrow R$  es un procedimiento tal que al ser consultado en un punto  $x \in R^n$  conocemos el valor de  $f$  y su gradiente  $\nabla f(x)$ .

La utilización de los oráculos para la optimización es algo muy conveniente, pues nos proporciona una forma de manipular la información saliente de un problema, o del gradiente del problema, siendo muy útil para la implementación de un algoritmo de entrenamiento.

Es importante tener en cuenta que los problemas de optimización tienen que responder dos cuestionamientos: a) **¿Que significa exactamente la palabra minimización?**, b) **¿Hay una forma eficiente de minimizar?**

Para la primera pregunta, debemos tener en cuenta que encontrar un mínimo exacto no siempre es posible, así muchas veces lo mejor que se puede hacer es aproximarse al mínimo. Esto da paso a la segunda pregunta, cuánto tiempo vamos a intentar obtener la aproximación, cuánto esfuerzo es el aceptable. En base a esto vamos a encontrar una estrategia de optimización que sea posible y con un esfuerzo razonable.

### 3.1. Descenso por el Gradiente.

Una de las estrategias de optimización usadas es la de descenso por gradiente. La idea básica es la siguiente:

Dada una función  $f(x)$ , la derivada  $f'(x)$  da la pendiente en el punto  $x$ , si la pendiente es positiva en ese punto la función es creciente y decreciente en caso contrario, si es cero no sabríamos con certeza si es máximo o mínimo, la pendiente es entonces un número negativo o positivo, si este número lo multiplicamos por un factor pequeño positivo  $\epsilon$ , tenemos que  $f(x - \epsilon f'(x)) \leq f(x)$ , esto es similar a cuando, por ejemplo, si estamos en el campo y vamos a iniciar la marcha, si notamos un desnivel positivo, sabemos que vamos hacia arriba, y si es negativo hacia abajo, y si no hay desnivel estamos en un lugar plano, esta es la razón por la que al método se le llama descenso por gradiente.

Al factor  $\epsilon$  se le llama tamaño de paso pues regula el tamaño del paso con el que vamos a decidir si asciende o desciende, esta variable tiene sus problemas pues si es muy pequeña el cálculo del mínimo puede ser muy tardado y si es muy grande puede ser errónea, para apreciar el descenso por la pendiente, es mejor ver el caso multivariado, pues en este caso, nos damos cuenta que puede haber varias direcciones que desciendan, pero no todas ellas podrían ser las que nos lleven al lugar que queremos. Nosotros vamos a utilizar **ADAM**.

### 3.2. ADAM.

**ADAM** es un método planteado en la “*International Conference on Learning Representations*” de 2015 y publicada hasta enero de 2017 por Diederik P. Kingma y Jimmy Lei Ba. Su propuesta es para mejorar el desempeño en la utilización del método de descenso por gradiente estocástico en funciones con gradiente de primer grado.

El método de gradiente estocástico utiliza lotes de datos para cada ciclo o época de entrenamiento, de cada lote se elige un elemento y sobre el conjunto de estas elecciones se actualizan los parámetros, dado que disminuye el número de datos sobre los que se realizan los cálculos disminuye, los requerimientos de memoria son menos.

**ADAM** hace más eficiente el método de descenso por gradiente estocástico, para funciones de gradientes de primer orden con pocos requerimientos de memoria. El método calcula las tasas de aprendizaje individual para diferentes parámetros basado en las estimaciones de los momentos de los gradientes (esta es la razón por la que el algoritmo se llama Adam-ADaptive Moment estimation). Combina las ventajas de dos métodos que fueron populares en ese tiempo: AdaGrad (Duchi et al., 2011), que trabaja competentemente con gradientes dispersos, y RMSProp (Tieleman & Hinton, 2012), que trabaja bien con parámetros no-estacionarios. Algunas de las ventajas más claras son que las magnitudes de las actualizaciones a los parámetros son invariantes al re-escalamiento del gradiente, el tamaño del avance está acotado por el tamaño del avance de los hiperparámetros.

En arquitecturas como la elegida para trabajo con redes neuronales convolucionales su rendimiento ha sido muy bueno, principalmente por su uso de memoria.

### 3.3 Función de pérdida.

Una vez establecido el método usado para actualizar los parámetros para optimizar nuestra función objetivo hay que tener una manera cuantitativa para determinar si nos acercamos al objetivo o es todo lo contrario. A esta función le llamamos función de pérdida, y es la que determina si el ejemplo procesado coincide con la etiqueta proporcionada y qué tanto coincide.

Para problemas de clasificación de varias etiquetas es común usar la función de Entropía Cruzada.

$$L = - \sum_{i=1}^N (y_i * \log(p_i)) \quad (18)$$

El algoritmo que usaremos para minimizar la entropía cruzada necesita dos parámetros; el logit que se está evaluando  $y = (y_1, ..y_n)$  y el logit con el que se compara  $p = (p_1, ..p_n)$ . Al primero lo llamaremos input logit, al segundo target logit. Lo que nos indicará la función es la cantidad de información esperada en dicha comparación, es decir, si hay diferencias grandes es que hay mucha información, y si el valor es pequeño pues la poca información nos indica que los logits son más parecidos.

## 4. Arquitecturas de Redes Neuronales.

En la mayoría de los trabajos para clasificación y reconocimiento de imágenes, la utilización de redes neuronales convolucionales es un estándar, aunque hay otros métodos de clasificación. En esta sección vamos a hablar brevemente de las arquitecturas más usadas y vamos a definir cuál es la que se adecúa mejor a nuestro objetivo.

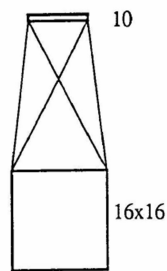
### 4.1. LeNet5 (1998).

En el artículo *“Generalization and network design strategies”* (1989)<sup>[13]</sup> de Yan Le Cun, **se** analizan las capacidades de clasificación de 5 implementaciones de redes neuronales, aplicadas al reconocimiento de caracteres (los 10 dígitos) en imágenes en escala gris de dimensión 32 x 32, todas aprendiendo con el algoritmo Back Propagation.

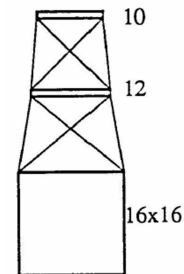
- **Net-1:** En su primera implementación utiliza una red neuronal de una sola capa completamente conectada con una salida de 10 nodos con la función sigmoide. El problema que se observa es que fácilmente se puede sobre entrenar, tiene un rendimiento decepcionante y “la clasificación se basa esencialmente en el cálculo de una superposición ponderada entre el patrón de entrada y ‘un prototipo promedio’”.
- **Net-2:** Se inserta una capa oculta de 12 unidades entre la primera capa y la capa de salida. El rendimiento mejora y es capaz de aprender en 6 épocas. Se observa un poco de sobre entrenamiento. Se observa que la desviación estándar sobre el rendimiento de entrenamiento es significativamente más grande que con la primera. Esta es una indicación que la red tiene bastante más indeterminación, y el número de soluciones consistentes es más grande. Sin embargo, en el conjunto de prueba no da los mismos resultados. *“De este resultado se concluye que la red es muy grande (o tiene muchos grados de libertad)”*.
- **Net-3:** Para evitar la generalización en la clasificación se reduce el tamaño de las capas de la red neuronal, Le Cun observa que para evitar la sobre parametrización de la red se pueden extraer características locales y combinarlas para formar una de grado más grande.

En su arquitectura propuesta para esta red propone dos capas; una H1 de tamaño 8 altura y 8 de ancho y otra H2 de tamaño 4 de altura y 4 ancho (para facilitar la lectura diremos de 8x8 o 4x4, según sea la dimensión de la capa). Esta reducción de características conserva activas las características significativas para la clasificación. Aunque el rendimiento no fue considerablemente más grande que con la red anterior, su ventaja estuvo en el costo computacional, pues Net-3 fue tres veces más pequeña que Net-2.

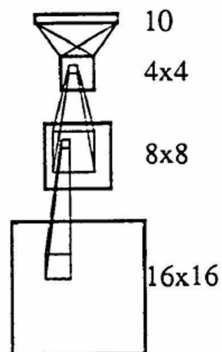
- **Net-4:** La arquitectura de Net-4 es bastante similar a Net-3, la primera capa H1 tiene dos filtros de 8x8 (el término usado es *feature map*, y corresponde a compactar segmentos del dato de entrada y conservar el más significativo, de esta manera se espera que las características más sobresalientes permanezcan activas). La segunda capa H2 es igual a la anterior. El rendimiento de ésta fue sobresaliente (94%) a pesar del número de sus pesos independientes.
- **Net-5:** Es la misma arquitectura que la anterior excepto que en sus capas tiene cuatro filtros de 4x4. Su rendimiento fue de 98% y facilitó el entrenamiento.



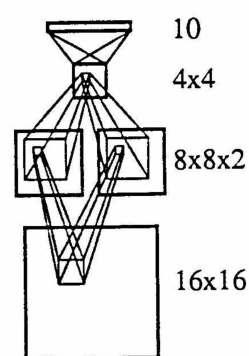
(a)



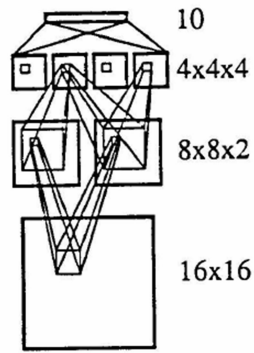
(b)



(c)



(d)



(e)

Figura 17. Arquitecturas de Net-1 (a), Net-2(b), Net-3(c), Net-4(d), y Net-5(e) (Imágenes extraídas del artículo de Le Cun<sup>[13]</sup>)

Fue hasta su artículo de 1990 “*Handwriting Digit Recognition with a Back-Propagation Network*” que Le Cun propone una arquitectura que utiliza las redes convolucionales con 5 capas, y en 1998 se implementa como hoy la conocemos. La arquitectura es como se presenta a continuación

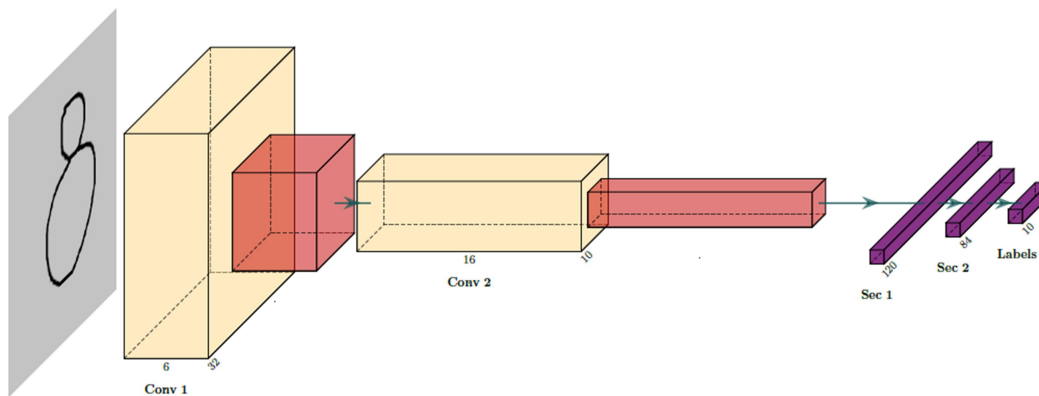


Figura 18. Arquitectura de LeNet 5.

Básicamente su funcionamiento es como sigue:

**Conv 1:** La primera capa convolucional aplica un filtro a la imagen dando como salida 6 ejemplos: los ejemplos son variaciones de la imagen original. El filtro usado en esta capa es de 5x5. Como resultado se tiene una salida de 28x28x6, es decir seis variaciones de 28x28 del original que era de 32x32.

**Maxpool 1:** Esta capa utiliza un filtro de 2x2 y un paso de dos para conservar el máximo valor del filtro y conservar la característica más relevante, al mismo tiempo reduce la dimensión de la salida a 14x14x6, de variaciones de 14x14.

**Conv 2:** Para esta capa se utiliza el mismo tamaño de filtro, es decir 5x5, para la salida el número de muestras aumenta a dieciséis, por lo que la salida final será de 10x10x16.

**Maxpool 2:** Con un filtro de 2x2 y un paso de dos, la salida ahora se reduce a 5x5x16.

**Sec 1:** Esta capa tiene 120 características y la capa de 5x5x16 se conecta completamente a ella. Los elementos de la salida de la capa anterior son aplanados en un vector unidimensional. Esta representación unidimensional conserva las características especiales para el aprendizaje de la red.

**Sec 2:** Las 120 celdas de la capa anterior se conectan a una capa de 84. Esta capa continua el proceso de destacar las características jerárquicas extraídas por la primera capa. Al igual que en la primera capa se utiliza una función sigmoide para introducir no linealidad (recordemos que de principio las redes neuronales fueron diseñadas para resolver problemas de clasificación que funciones lineales no podían resolver, como el caso de la función booleana XOR ) a la red, además de facilitar el aprendizaje y facilitar el flujo de información.

**Labels:** Es la capa de salida y tiene el número de clases de salida del proceso de clasificación. Por lo general en esta capa se utiliza la función de activación softmax y el resultado es un logit, que nos indica la respuesta más probable a partir de la entrada.

## 4.2. Alexnet (2012).

Comprende 8 capas, las primeras 5 son capas convolucionales y las ultimas 3 son secuenciales. Esta arquitectura es propuesta por Alex Krizhevsky e Ilya Sutskever<sup>[11]</sup>, ellos proponen solapar los filtros de la capa de Maxpooling y les llama así “Maxpooling solapado”. La red fue sometida al concurso ImageNet LSVRC-2012, en donde el reto es clasificar 1000 diferentes etiquetas y un conjunto de imágenes de más de 14 millones, es de las primeras redes en implementar el uso de GPUs (desde su primera versión de 2010) para el procesamiento, quedando en primer lugar con una tasa de error de 15%, siendo menor en comparación con el segundo lugar que tuvo una tasa de error de

26.2%. En la Figura 19 vemos la imagen de la arquitectura proporcionada por Krihzevky et al en su artículo de 2012 “ImageNet Classification with Deep Convolutional Neural Networks”<sup>[11]</sup>.

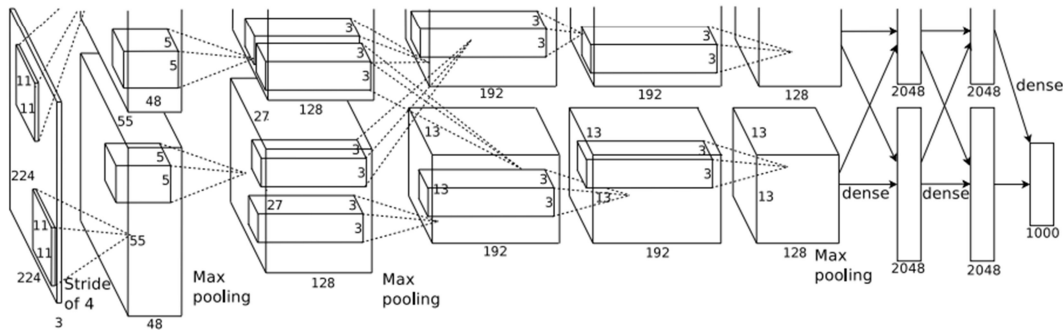


Figura 19. Arquitectura de Alexnet. Tomada del artículo “ImageNet Classification with Deep Convolutional Neural Networks”<sup>[11]</sup>. Muestra la delineación de responsabilidades de dos GPU’s, una GPU procesa las partes superiores de la imagen y la otra la inferior. La comunicación entre ambas solo se da en algunas capas.

En donde, lo interesante de su figura es que explican claramente las responsabilidades de procesamiento de cada GPU.

Una de las características más importantes de esta red es que son 60 millones de parámetros y 650,000 neuronas.

Alexnet utiliza Descenso por gradiente estocástico. Krihznevsky aumentó los datos para evitar el sobreajuste, le tomo seis días el entrenamiento en dos GTX580 de 3 GB a 90 ciclos.

#### 4.3. VGG16 ( 2014 ).

VGG16 ganó el ImageNet LSVRC-2014, su tasa de error fue de 7.3%. Tuvo varias las configuraciones que propone en su artículo, estas son la A, A\_LRN, B, C, D, E, su única diferencia es la profundidad, como podemos ver en la figura 20.

| ConvNet Configuration       |                        |                               |                                            |                                            |                                                         |
|-----------------------------|------------------------|-------------------------------|--------------------------------------------|--------------------------------------------|---------------------------------------------------------|
| A                           | A-LRN                  | B                             | C                                          | D                                          | E                                                       |
| 11 weight layers            | 11 weight layers       | 13 weight layers              | 16 weight layers                           | 16 weight layers                           | 19 weight layers                                        |
| input (224 × 224 RGB image) |                        |                               |                                            |                                            |                                                         |
| conv3-64                    | conv3-64<br><b>LRN</b> | conv3-64<br><b>conv3-64</b>   | conv3-64<br>conv3-64                       | conv3-64<br>conv3-64                       | conv3-64<br>conv3-64                                    |
| maxpool                     |                        |                               |                                            |                                            |                                                         |
| conv3-128                   | conv3-128              | conv3-128<br><b>conv3-128</b> | conv3-128<br>conv3-128                     | conv3-128<br>conv3-128                     | conv3-128<br>conv3-128                                  |
| maxpool                     |                        |                               |                                            |                                            |                                                         |
| conv3-256<br>conv3-256      | conv3-256<br>conv3-256 | conv3-256<br>conv3-256        | conv3-256<br>conv3-256<br><b>conv1-256</b> | conv3-256<br>conv3-256<br><b>conv3-256</b> | conv3-256<br>conv3-256<br>conv3-256<br><b>conv3-256</b> |
| maxpool                     |                        |                               |                                            |                                            |                                                         |
| conv3-512<br>conv3-512      | conv3-512<br>conv3-512 | conv3-512<br>conv3-512        | conv3-512<br>conv3-512<br><b>conv1-512</b> | conv3-512<br>conv3-512<br><b>conv3-512</b> | conv3-512<br>conv3-512<br>conv3-512<br><b>conv3-512</b> |
| maxpool                     |                        |                               |                                            |                                            |                                                         |
| conv3-512<br>conv3-512      | conv3-512<br>conv3-512 | conv3-512<br>conv3-512        | conv3-512<br>conv3-512<br><b>conv1-512</b> | conv3-512<br>conv3-512<br><b>conv3-512</b> | conv3-512<br>conv3-512<br>conv3-512<br><b>conv3-512</b> |
| maxpool                     |                        |                               |                                            |                                            |                                                         |
| FC-4096                     |                        |                               |                                            |                                            |                                                         |
| FC-4096                     |                        |                               |                                            |                                            |                                                         |
| FC-1000                     |                        |                               |                                            |                                            |                                                         |
| soft-max                    |                        |                               |                                            |                                            |                                                         |

Figura 20. Tabla con las arquitecturas de VGG16. Obtenida del artículo de Karen Simonyan & Andrew Zisserman “Very Deep Convolutional Networks for Large-Scale Image Recognition Vol 12 Issue 08”<sup>[24]</sup>

**VGG16** necesita de un pre procesamiento, que consiste en restar a cada píxel el valor RGB medio, calculado en el conjunto de entrenamiento., y consta de 138 millones de parámetros.

El problema que tiene este modelo es que entrenarlo lleva mucho tiempo, en el artículo especifica: “en un sistema equipado con cuatro GPUs NVIDIA Titan Black, el entrenamiento de una sola red tomó 2–3 semanas dependiendo de la arquitectura”.

Para solventar este problema las principales librerías de redes neuronales (**Tensorflow** , **Keras**, **Pytorch**) tienen modelos entrenados. En la actualidad hay dos modelos disponibles en internet **VGG16** y **VGG19**.

#### 4.4. *GoogleNet* (2014).

Fue desarrollada en Google, posteriormente conocida como Inception. En su artículo Christian Szegedy et al (septiembre 2014), “*Going deeper with convolutions*”<sup>[25]</sup>. Su nivel de tasa de error fue de 6.67%, que según el artículo de Olga Russakovsky et al “*ImageNet Large Scale Visual Recognition Challenge*” la tasa de error en humanos para el mismo set fue de 5.8%. La arquitectura básica es de 22 capas convolucionales y 27 capas maxpooling, dando como resultado 4 millones de parámetros. Una de las características más interesantes de la arquitectura es el uso de pooling paralelo en las capas de maxpooling.

#### 4.5 *ResNet* (2015).

**ResNet** utiliza el concepto de *Residual Learning*, que es básicamente que la entrada de una red siga su camino pero que a la vez salte algunas capas y se sume a la salida de ese segmento. Fue la primera red convolucional en superar la tasa de error del humano y tener una complejidad de **parámetros** inferior a **VGGNet**.

#### 4.6. Otras arquitecturas.

A partir de 2016 el desarrollo de arquitecturas se incrementó en número de capas y complejidad debido al número de parámetros. Para el objeto de este trabajo no considero que sean de importancia pues nuestros datos son de una sola dimensión y de poca complejidad. Solo añadiremos los trabajos más importantes de los últimos años por si alguien está interesado: SqueezeNet (2016- Iandola et al), ResNext (2017 – Xie et al), MnasNet (2018 – Tan et al), ShuffleNet (2017 – Zhang et al).

#### 4.7. Arquitectura elegida.

Dado que los datos que vamos a procesar se pueden acomodar en una red convolucional de una sola dimensión vamos a usar como base de nuestra arquitectura LeNet5. De esta arquitectura sabemos que es muy robusta y da muy buenos resultados en el procesamiento de señales.

## 5. Desarrollo Práctico.

### 5.1. Imágenes Hiperespectrales.

La luz es algo muy natural en nuestra vida y nuestros ancestros daban muchas cosas por hecho de ella. Sin embargo, el verdadero inicio del estudio de la luz y sus propiedades se podría remontar al experimento de Sir Isaac Newton con un prisma de cristal al ver el paso de la luz por el prisma y como al otro lado del prisma se revelaban siete colores. Al principio se le atribuyó ese efecto a las impurezas del cristal, pero al poner otro prisma y recuperar el estado del rayo de luz original se empezaron a dar cuenta que la luz contenía algo más. Pero, ¿cuáles son sus verdaderas propiedades?, ¿es una onda?, ¿es una partícula? Sus propiedades como partícula habían sido estudiadas desde los griegos, en la reflexión y refracción se trataba como una partícula, y tratándola como partícula fueron capaces de usar la luz. Fue en el siglo XIX en el experimento de Thomas Young de la doble rendija, que se muestra como una onda, y pudo observar patrones de interferencia. Por la misma fecha William Herschel observó que los colores tienen diferente temperatura, el color azul es más frío que el color rojo, de hecho se dio cuenta que la energía emitida más allá del color rojo es mucho más caliente, a esta zona se le llamo región infrarroja.

La relación entre estos descubrimientos que interrelacionaron el color, frecuencia y ancho de onda, son los que dieron un marco de referencia a detección remota de imágenes hiperespectrales, ya que pueden ser usados para caracterizar la reflexión de la luz contra objetos.

Otro aporte importante se dio con el avance de la industria aeroespacial, el poder reconocer territorios vastos desde el espacio, introdujo la necesidad de obtener información cualitativa de estos territorios, siendo el Departamento de Defensa de los estados Unidos(DoD) y la Administración Nacional Aeronáutica y del Espacio (NASA), quienes han patrocinado muchos proyectos de reconocimiento geográfico. De igual manera los programas espaciales Franceses y Japoneses han participado en este tipo de iniciativas. Los primeros dispositivos eran cámaras fotográficas normales, sin embargo, en la década de los 60's el DoD **es** el que implementa las primeras cámaras con películas infrarrojas con filtros especiales en sus aviones de reconocimiento U2. Estas cámaras son reconocidas como las primeras cámaras de tecnología multiespectral. En la década de los 70's la NASA lanza sus satélites Landsat. Estos sistemas fueron diseñados para obtener información multiespectral de la superficie de la tierra, fueron los primeros sistemas en codificar digitalmente la información obtenida.

Antes de seguir hagamos algunas precisiones en el ámbito del estudio de la imagenología Hiperespectral; la materia a una temperatura por arriba del cero absoluto irradia energía electromagnética, la cantidad de energía radiada de un objeto depende de su temperatura absoluta de emisividad y es una función de la longitud de onda. Al rango total de la longitud de onda se le conoce como Espectro Electromagnético (Figura 21).

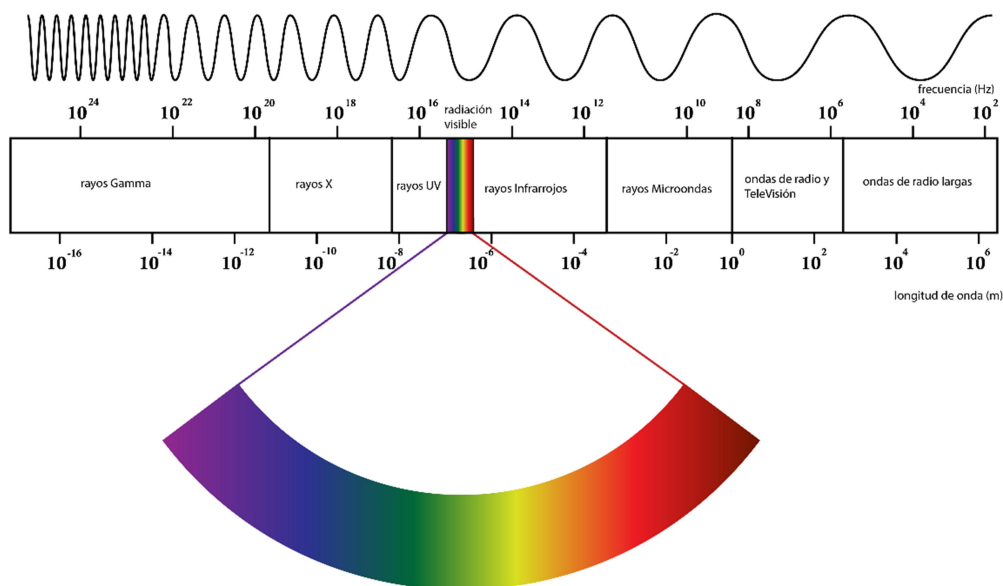


Figura 21. Espectro electromagnético (Figura obtenida del sitio de la Unidad de Apoyo al Aprendizaje de la Facultad de Arquitectura. [https://uapa.cuaieed.unam.mx/sites/default/files/minisite/static/940d57bb-b822-4ca5-8394-2fcc4e68e5e2/principios\\_propagacion\\_luz\\_calor\\_sonido/img/img\\_contenido/8\\_espectro.png](https://uapa.cuaieed.unam.mx/sites/default/files/minisite/static/940d57bb-b822-4ca5-8394-2fcc4e68e5e2/principios_propagacion_luz_calor_sonido/img/img_contenido/8_espectro.png) )

Las imágenes hiperespectrales almacenan su información en un arreglo de tres dimensiones frecuentemente llamado cubo, en el que las primeras dos entradas corresponden a coordenadas espaciales de la imagen (alto y ancho) y la tercera es el contenido espectral. Al contenido espectral lo referenciaremos como bandas, y al contenido espacial trataremos como a cualquier imagen y lo trataremos como un píxel de cualquier imagen. La información contenida en la banda depende mucho del método de adquisición de la información. Diremos que una imagen es Hiperespectral si el número de bandas va de más de una decena a cientos de estas. Si el número de bandas es de tres a diez llamaremos a la imagen multispectral.

## 5.2. Espectros para clasificación.

El objetivo de este trabajo es la clasificación de imágenes Hiperespectrales, en el trabajo base se tomaron imágenes con 128 bandas espectrales, es decir cada píxel de la imagen puede verse como un histograma de 128 datos, este histograma es específico de cada pigmento, la variación en el histograma se debe a mezclas o diferencias de iluminación, sin embargo, la estructura básica del pigmento es razonablemente distinguible.

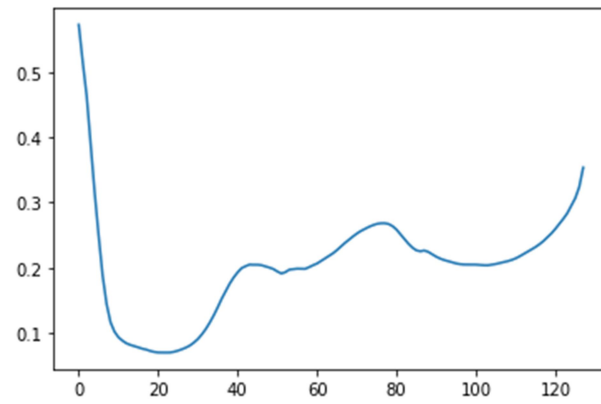


Figura 22. Ejemplo de histograma de un píxel.

El primer objetivo será clasificar una serie de pinturas tomando como base una paleta de 25 colores (Figura 23).



Figura 23. Paleta de Pigmentos en color original.

El formato en que la imagen se procesa es el ENVI file format. Este formato consta de dos archivos, un archivo con extensión hdr, que tiene el encabezado de la imagen en formato ASCII, describiendo básicamente las dimensiones de la imagen (description, samples, lines, bands, etc.) y el archivo con la información espacial de extensión dat, que es un vaciado de bytes en uno de los tres formatos siguientes: **Band Sequential (BSQ)**, **Band-interleaved-by-pixel (BIP)** y **Band-interleaved-by-line (BIL)**. Estos formatos vienen especificados en el campo “interleave” del encabezado. Para nosotros no es de importancia el tipo interleave, pues la librería del programa de clasificación lo maneja sin intervención de ningún parámetro en la llamada a la función de obtención de imagen.

Cada cuadro en la imagen es un pigmento, tomamos cada pigmento como un cuadrado de 110 por 110 píxeles y con eso se abarca el pigmento en su totalidad, el fondo de la tabla no se tomará en cuenta (Figura 24).

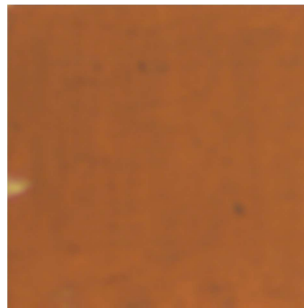


Figura 24. Pigmento 1 (110x110 píxeles).

Cada pigmento tiene aproximadamente 12,100 píxeles, cada píxel es un histograma de 128 entradas.

### 5.3. Procesamiento de Datos.

Para el conjunto de entrenamiento, se tomó la región del pigmento de  $110 \times 100$  píxeles, para el primer pigmento queda distribuida más o menos como en la Figura 25 :

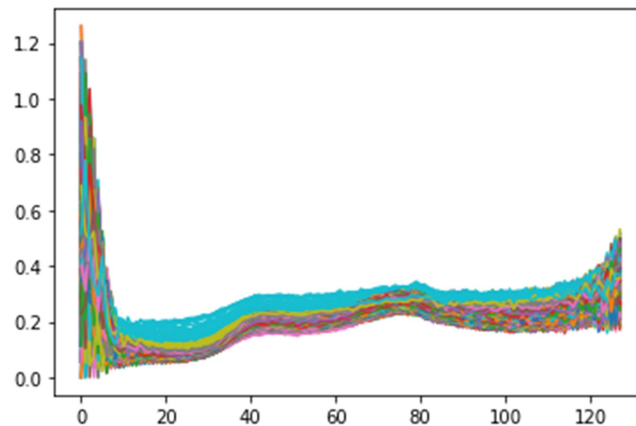


Figura 25. Pigmentos de una zona elegida.

Se podría diferenciar una zona azul que probablemente corresponda a zonas con más iluminación, en donde, para poder diferenciar de una mejor manera, aplicamos el algoritmo k-means para obtener varios centroides y ver qué tan diferentes podrían ser los espectros del primer pigmento.

En la Figura 26. tenemos el primer pigmento separado en 10 centroides.

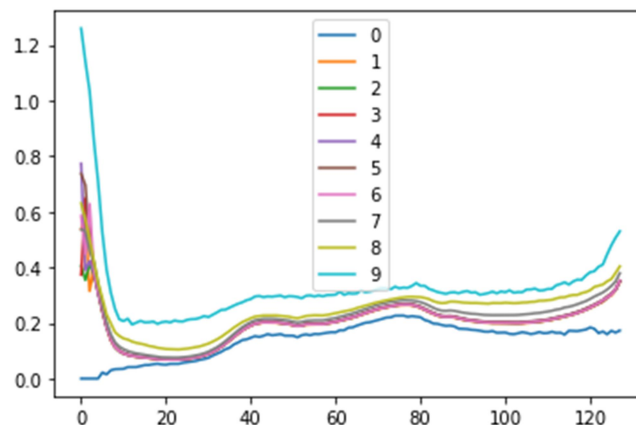


Figura 26. Histograma de pigmento con 10 centroides.

Como se puede ver no es necesario tener 10 centroides, los centroides que no siguen el patrón del pigmento son el centroide 0 y el centroide 9, en la siguiente figura con 5 centroides podremos distinguir mejor esta situación como lo muestra la Figura 27.

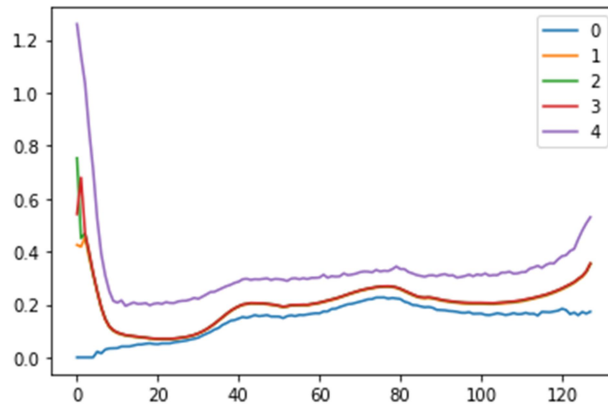


Figura 27. Histogramas de pigmentos con 5 centroides.

Podemos ver que el centroide 0 y el 4 son los que no se ajustan fácilmente al patrón del pigmento, la principal diferencia se da al principio y al final del histograma.

En la Figura 28. veremos cómo están distribuidos en el rectángulo.

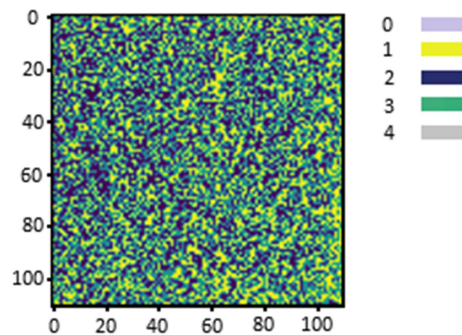


Figura 28. Imagen de zona de pigmentos, coloreado con identificación de centroides.

Se distinguen básicamente tres centroides, el centroide 0 y el 4 casi no tienen representantes. En general la mayoría de los píxeles quedan en las primeras tres categorías y los píxeles que no están en esas son los que están siendo afectados por la iluminación a favor o en contra. Para trabajar con esto vamos a tomar las bandas a partir de la banda 9 hasta la 120.

Dependiendo de la región elegida, se generan el número de elementos para el entrenamiento. En un rectángulo de 110 x 110 se tiene a lo más 12,100 muestras para entrenamiento.

#### 5.4. Entrenamiento y Clasificación de la red neuronal.

Para el proceso de entrenamiento y clasificación tomaremos la paleta de 25 colores y cada pigmento de 110 x 110 numerados como en la figura 20, probamos con tres escenarios diferentes: 2,000 muestras por etiqueta para el primer entrenamiento, 12,100 muestras por píxel para el segundo entrenamiento y, 12,100 muestras por píxel para el tercer entrenamiento pero seleccionando 112 bandas de cada píxel. Las bandas que se eligieron son la que resultan de quitar las 8 primeras bandas y las 8 últimas, esta consideración se hizo para disminuir el ruido incorporado por el instrumento de captura de imágenes.

La arquitectura usada esta basada en LeNet-5, y se usamuchu en el procesamiento de señales. A continuación se muestr en la figura 29 su arquitectura.

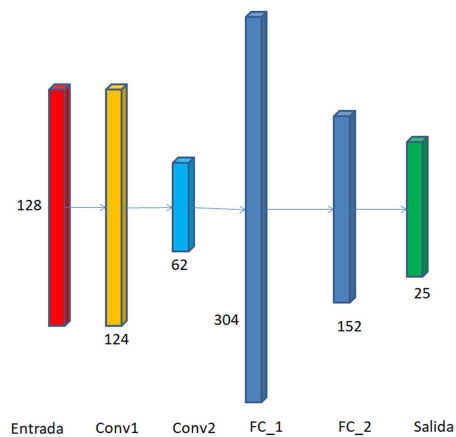


Figura 29. Arquitectura usada para la clasificación de los pigmentos. Son 5 capas con una entrada de un histograma con la información de un píxel de la imagen espectral y una salida con la clasificación de una de las 25 clases de colores.

#### 5.4.1. Primera red entrenada.

La primera configuración fue: un conjunto de entrenamiento de 50,000 muestras (2,000 píxeles por cada uno de los 25 rectángulos de 110x100), 40,000 fueron usadas para el entrenamiento y 10,000 para la prueba. El número de épocas elegido fue 10 pues la convergencia era muy rápida como para usar un número más alto (Figura 30):

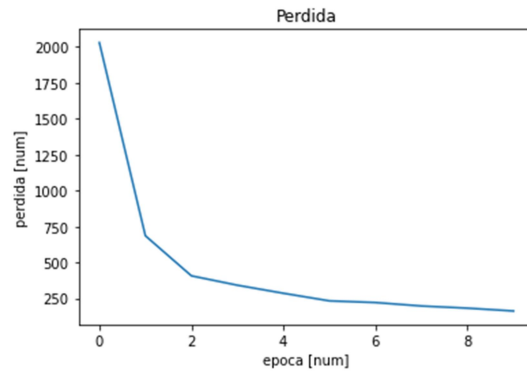


Figura 30. Curva de convergencia de entrenamiento de la primera red.

Para visualizar el proceso de clasificación tomamos un píxel al azar y vimos la forma en que lo procesa este modelo entrenado:

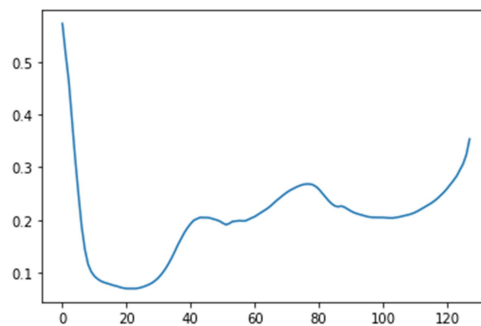


Figura 30. Entrada un píxel: 1x128.

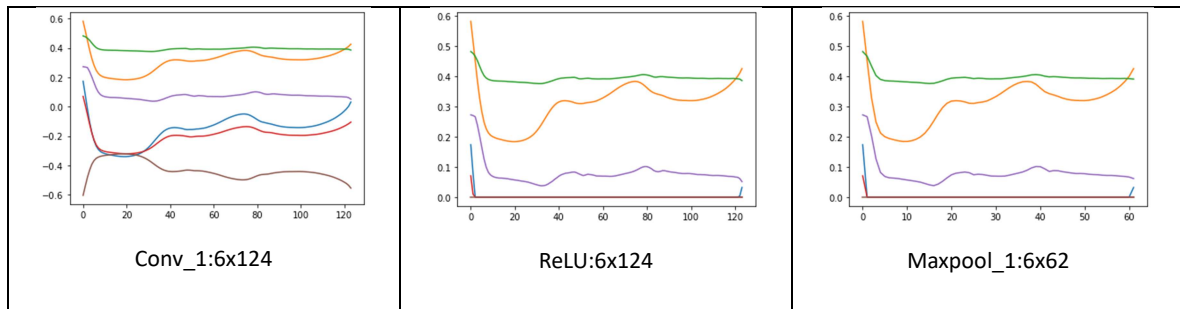


Figura 32. Capa convolucional Conv1.

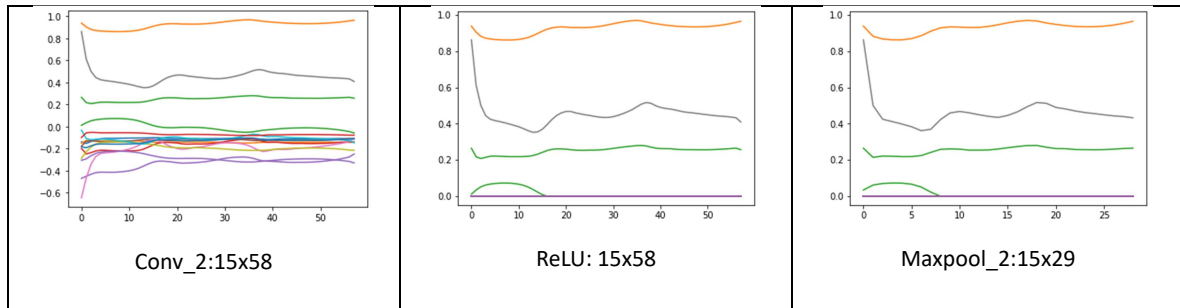


Figura 33. Capa convolucional Conv2.

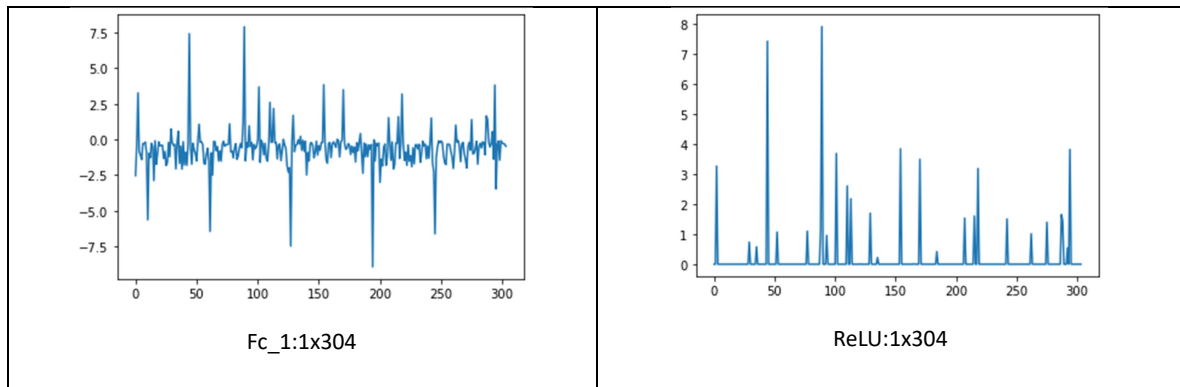


Figura 34. Capa lineal FC\_1.

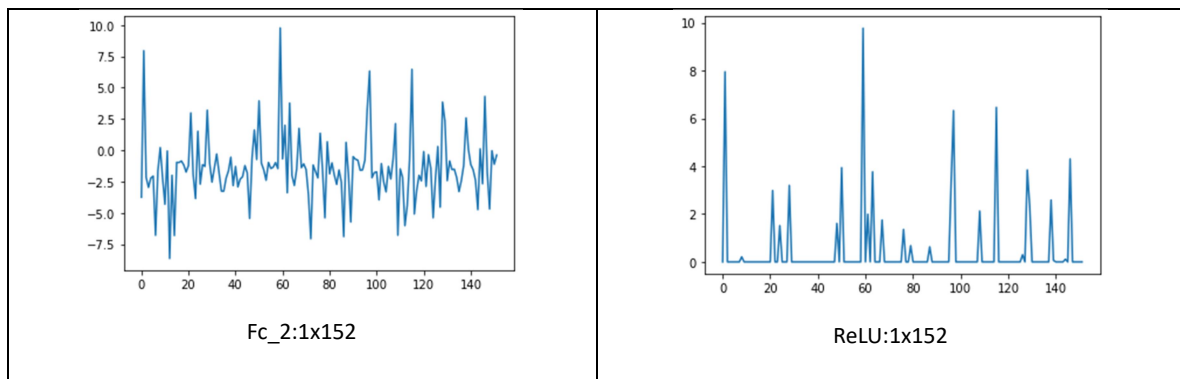


Figura 35. Capa lineal FC\_2.

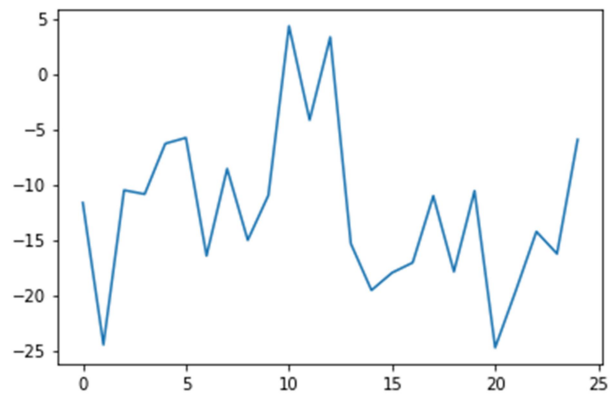


Figura 36. Salida resultado de la clasificación 1x25.

El resultado es un vector de 25 entradas, cada una corresponde a una probabilidad de pertenecer a una etiqueta de los 25 pigmentos de la paleta, y se toma el más probable, a este vector se le llama logit, y cada pigmento tiene su forma específica donde hay una característica específica para cada una. A continuación, se muestra la clasificación de la tabla de 25 pigmentos, a la izquierda esta la original y a la derecha la tabla clasificada, en la que vemos varias zonas que no están bien determinadas principalmente en los pigmentos 21 y 26 (Figura 36).

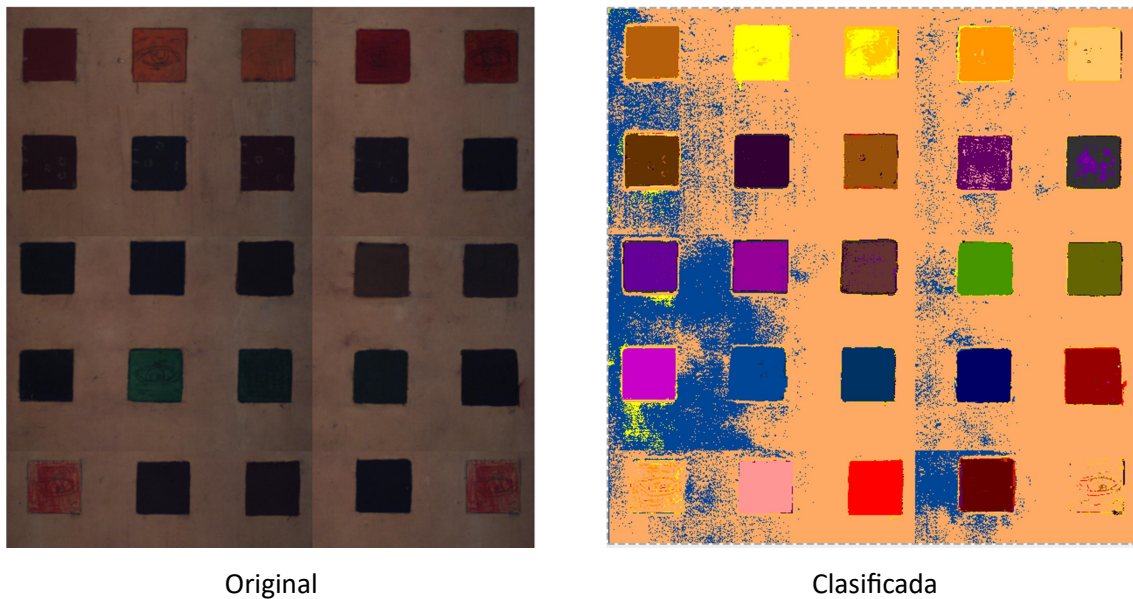


Figura 37. Paleta de entrenamiento procesada por la red neuronal entrenada.

#### 5.4.2 Segunda red entrenada.

Para la segunda red se aumentó el número de muestras a 12,100 por pigmento (todo el rectángulo del pigmento) y se usaron las 128 bandas. En la curva de perdida contra época se ve que la convergencia es muy rápida. El número de épocas parece adecuado y el tiempo de procesamiento apenas fue de tres horas en el equipo especificado en el Apéndice 1, tomando en cuenta que fueron más de 240,000 elementos para el conjunto de entrenamiento (Figura 37).

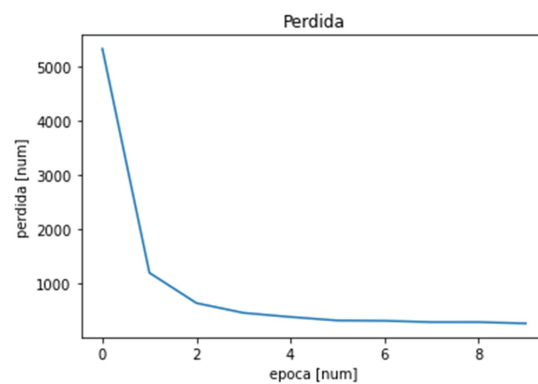


Figura 38. Resultado de la segunda red entrenada.

La clasificación en los pigmentos mejoro notablemente sin embargo el problema en la confusión de algunos pigmentos con el fondo se sigue presentando. Este problema es natural, pues, el fondo tiene un espectro que en algún grado es parecido a los pigmentos con los que son confundidos.

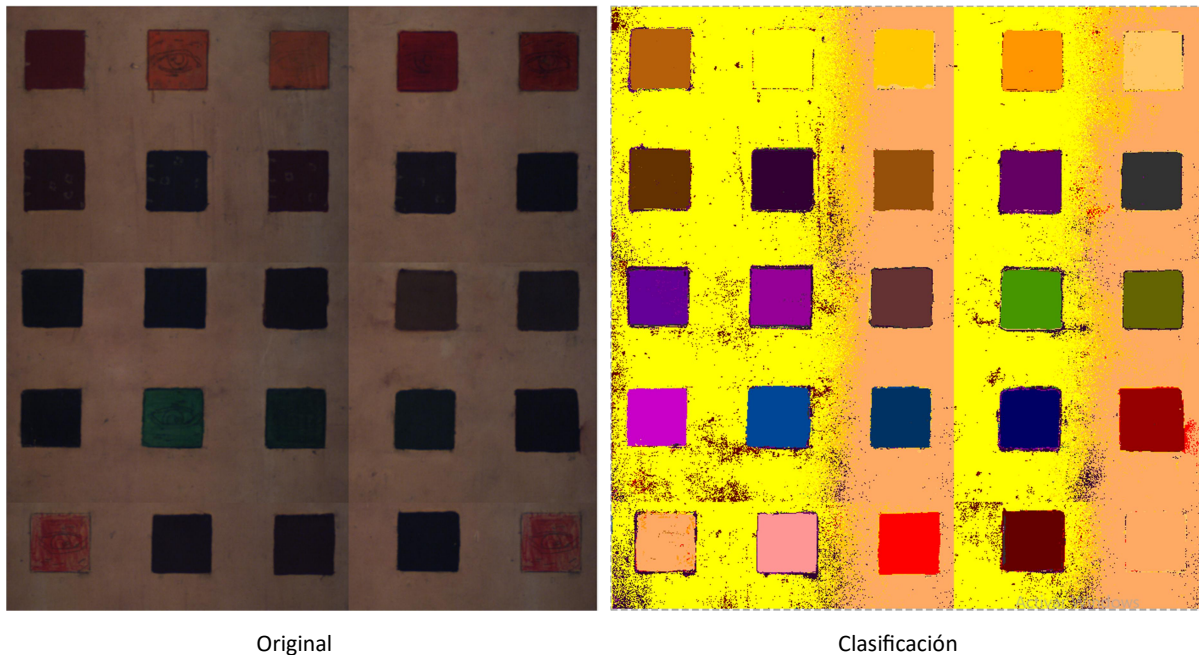


Figura 39. Resultado en imagen de la segunda red entrenada.

#### 5.4.3 Tercera red entrenada.

Para ésta se toman los rectángulos de 110x100 pixeles de los 25 pigmentos, pero se eliminan las ocho primeras bandas y las últimas ocho, con el mismo conjunto de entrenamiento, su convergencia no es muy diferente pero su funcionamiento en clasificación si es notable (Figura 39).

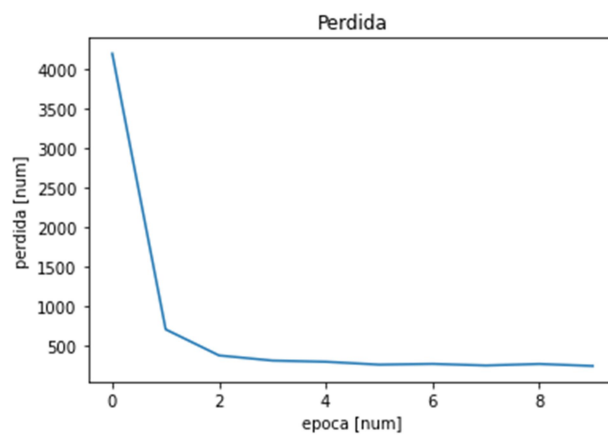


Figura 40. Resultado de la tercera red entrenada.

En esta clasificación la mejora se notó en los bordes de los pigmentos y en la uniformidad de la clasificación, aún persiste el problema de la confusión del fondo, pero este problema se resolverá al establecer un método de comparación de salidas (logits) para descartar los pigmentos que arrojen una probabilidad más cercana con el pigmento, pero no tengan la misma estructura en las demás etiquetas. Esto lo abordaremos con más profundidad posteriormente (Figura 40).

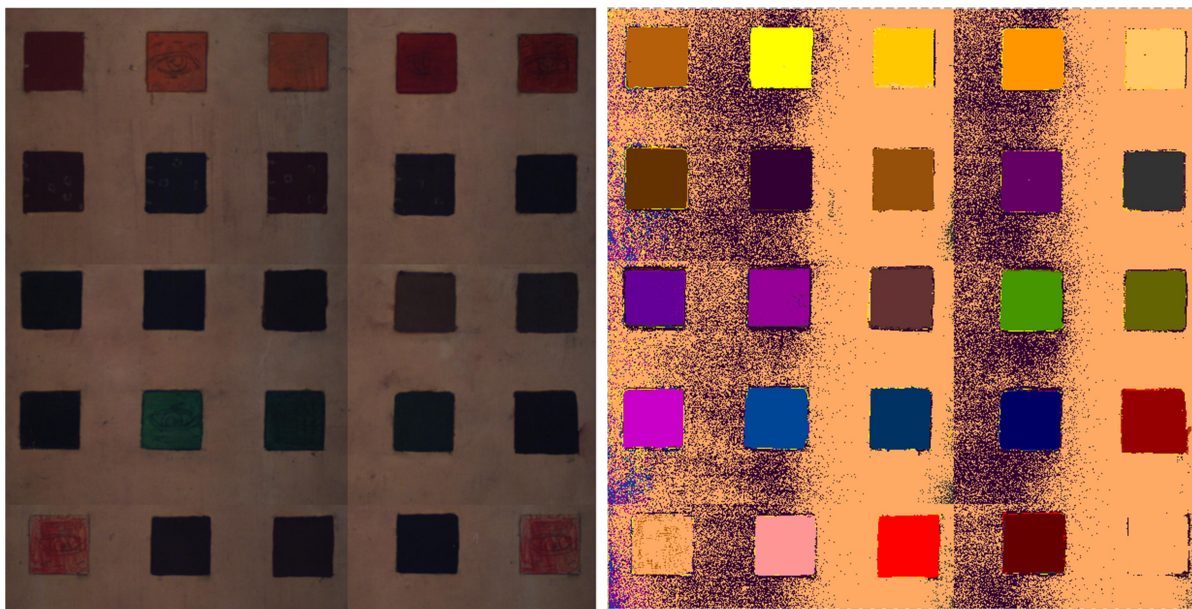


Figura 41. Resultado en imagen de la tercera red entrenada.

A pesar que hay zonas distinguibles con los pigmentos, toda el área fue identificada con algún color de la paleta, esto se debe a que todos los pigmentos van a ser clasificados, así, si de las 25 etiquetas, hay una a la que se le parece más, esa será su clasificación, independientemente de si el histograma pertenece a la etiqueta del pigmento o no. Para mejorar esta situación hay que encontrar una forma de medir la “distancia” entre el pigmento “ideal” y el pigmento clasificado. Encontrar un pigmento ideal no es sencillo, así que lo elegiremos de otra manera.

En el proceso de entrenamiento, cada región que fue etiquetada, tenía, histogramas mayormente de un pigmento, con más iluminación o con menos iluminación y además había algunas impurezas, esto modifica la forma de cada histograma de la región etiquetada, así al usar el algoritmo k-means en una región etiquetada obtendremos diferentes centroides, como se muestra en la Figura 24. Al

contar el número de pixeles cercanos a cada centroide se obtuvo en el pigmento 1 la siguiente tabla:

| N° de centroide | Cantidad de puntos |
|-----------------|--------------------|
| 1               | 0                  |
| 5               | 0                  |
| 3               | 3764               |
| 4               | 3935               |
| 2               | 4401               |
| Total           | 12100              |

Tabla 3. Pixeles cercanos a cada centroide correspondiente a la tercera red entrenada.

La mayoría de los pigmentos que va a ser etiquetados como del primer pigmento caen en tres centroides. Son estos puntos los que serán tomados en cuenta para obtener un único histograma que los represente y tenga una cercanía más precisa al pigmento número 1. De este histograma se va a obtener el logit target que se utilizara para comparar con el input logit y tener un parámetro de acercamiento entre estos. A continuación se elegirá el umbral que deben de cumplir para determinar que tan cerca los logits están para ser considerados etiquetados de una clase.

El umbral sera un múltiplo de la desviación estándar de los input logits de cada sección de pigmento que se usó para entrenar.

Paleta Original



Figura 42.

Tomando 1.5 veces la desviación estándar de los logits que fueron clasificados con los datos de entrenamiento.

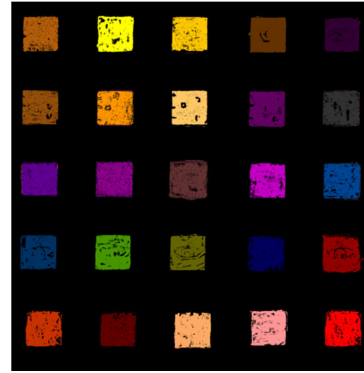


Figura 43.

Tomando 2 veces la desviación estándar de los logits que fueron clasificados con los datos de entrenamiento.

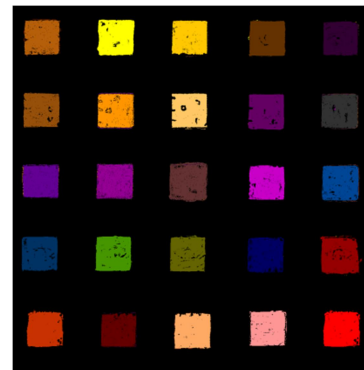


Figura 44.

Tomando 3 veces la desviación estándar de los logits que fueron clasificados con los datos de entrenamiento.

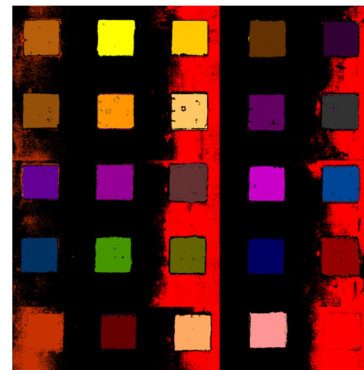


Figura 45.

El mejor resultado fue obtenido con un umbral 2 veces la desviación estándar, así lo probamos con las imágenes del pentecostés del museo de San Carlos y sin umbral se obtuvo la siguiente imagen:

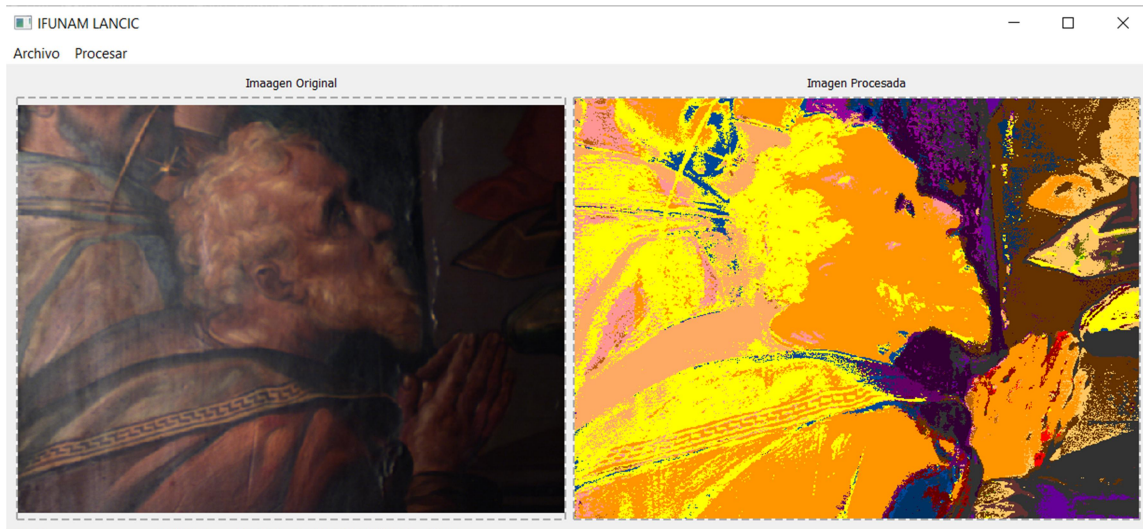


Figura 46. Imagen obtenida tomando dos magnitudes la desviación estándar para la clasificación.

Pero con el umbral se tuvieron los siguientes resultados:

Imagen con 1.5 veces la desviación como umbral de clasificación

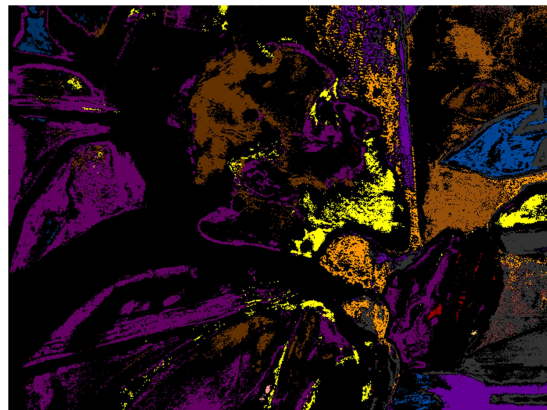


Figura 47.

Imagen con 2 veces la desviación como umbral de clasificación

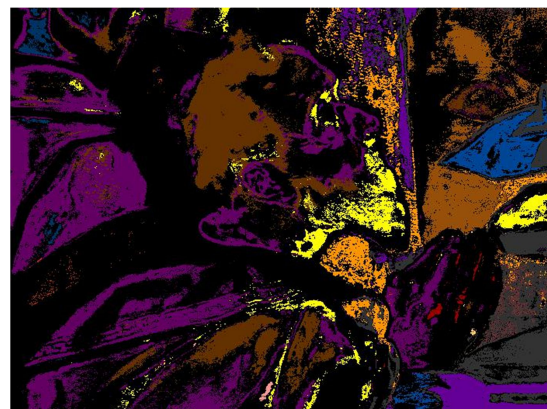


Figura 48.

Imagen con 3 veces la desviación como umbral de clasificación

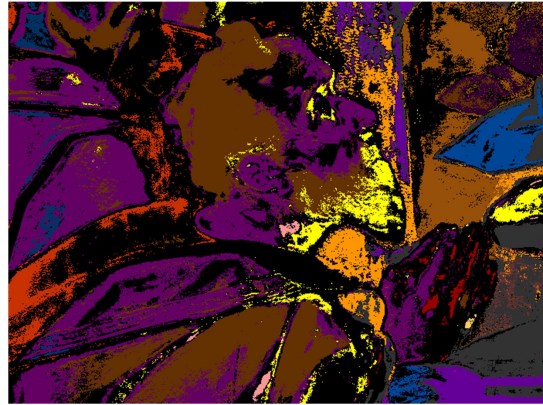


Figura 49.

Para comparar el funcionamiento de la clasificación, se utilizaron imágenes hiperespectrales del mural de Diego Rivera *Sueño de una tarde dominical en la Alameda Central*. De éste, se obtuvieron varias imágenes de diferentes lugares y que están indicados con rojo en la Figura 46. La imagen hiperespectral al igual que el modelo anterior es de 128 bandas.



Figura 50. Diego Rivera, *Sueño de una tarde dominical en la Alameda Central*, 1947, 4.8 x 15 m (Museo Mural Diego Rivera, originalmente, Hotel del Prado, ciudad de Mexico; foto: [Fedaro](https://www.fedaro.com), CC BY-SA 4.0)

[https://upload.wikimedia.org/wikipedia/commons/2/20/Mural\\_Sue%C3%B1o\\_de\\_una\\_tarde\\_dominical\\_en\\_la\\_Alameda\\_Central.jpg](https://upload.wikimedia.org/wikipedia/commons/2/20/Mural_Sue%C3%B1o_de_una_tarde_dominical_en_la_Alameda_Central.jpg)

Para la clasificación se utilizó una paleta de 20 pigmentos para generar el modelo con la misma arquitectura de LeNet5, pero hay una serie de probetas que contienen una gama más amplia de materiales que se utilizaran posteriormente para su clasificación, por el momento solo se procesaron con dicha paleta (Figura 47):



Figura 51. Paleta original de 20 pigmentos.

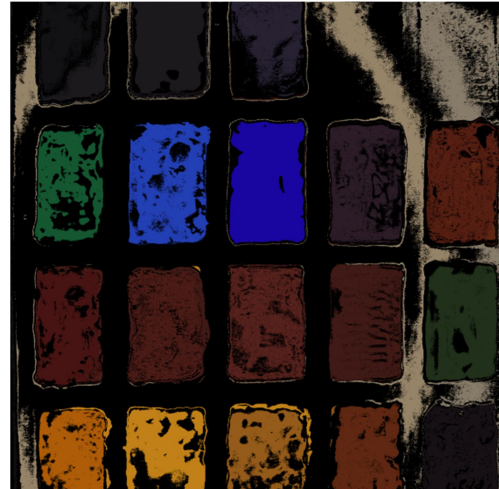


Figura 52. Clasificación con 2 veces la desviación estándar.

### Imágenes Clasificadas:



Figura 53. Fragmento de imagen del mural.



Figura 54. Imagen procesada.



Figura 55. Fragmento de imagen del mural

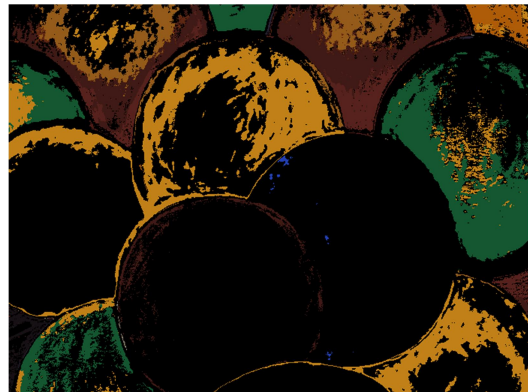


Figura 56. Imagen procesada

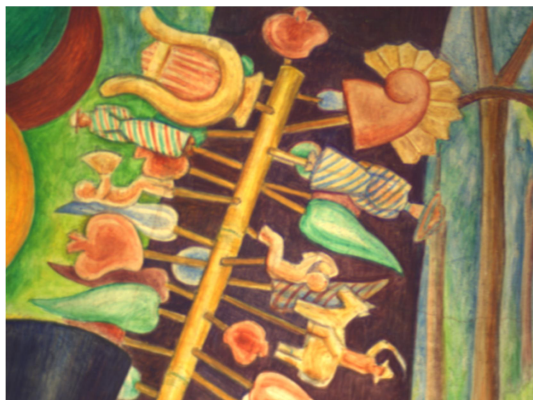


Figura 57. Fragmento de imagen del mural

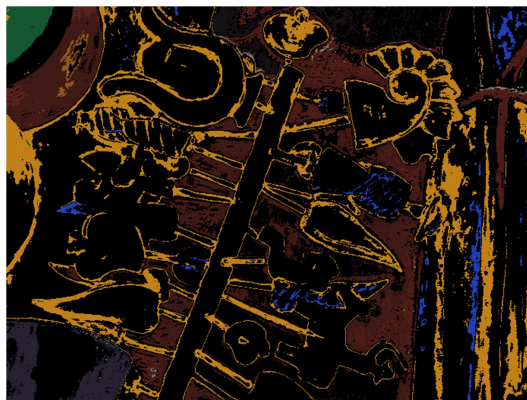


Figura 58. Imagen procesada



Figura 59. Fragmento de imagen del mural



Figura 60. Imagen procesada

## 6. Conclusiones.

La utilización de una red neuronal convolucional unidimensional, puede ser útil para determinar el tipo de pigmentos que se encuentran en las obras del patrimonio cultural, sin embargo es necesario que se tengan registrados los pigmentos y materiales que fueron utilizados, y no solo se puede limitar al acervo pictórico sino también al acervo de piezas de las que se pueda obtener información hiperespectral. La precisión en la clasificación dependerá de las paletas y clases base con las que se entrenen tales redes neuronales, sin embargo es claro que puede facilitar el trabajo de clasificación para el investigador.

## Apéndices.

### Apéndice 1. Infraestructura.

#### A.1. Hardware

La infraestructura proporcionada es la siguiente:

Equipo: Dell Precisión T5400

Procesador - Procesador Dual-Core Intel® Xeon® Serie 5200, E5405 @ 2.00GHz

Memoria – 5 GB SDRAM DDR2 con búfer completo a 667 MHz con 8 slots

Disco Duro - primario IDE 1 TB (útil 932), secundario Disco de estado sólido 1 TB (útil 932)

Tarjeta Gráfica – NVIDIA GeForce GTX 750 Ti, 640 núcleos CUDA

Sistema Operativo: Windows 10 Profesional Edition

Se adquirieron 4 dims de 2 GB y se instalaron para aumentar la memoria del equipo a 13 GB. El ambiente de programación es con Python y Pytorch, para aprovechar el componente de desarrollo CUDA y mejorar el rendimiento de los procesos.

#### A.1.2 Software.

Para el desarrollo se tuvieron que instalar varios componentes.

##### A.1.2.1. CUDA.

Para poder utilizar los núcleos de la tarjeta gráfica, es necesario instalar el CUDA Toolkit de NVIDIA. La versión correcta a instalar debe revisarse en la página de instalación de Pytorch para que tengamos una versión compatible. En nuestro caso usaremos la versión 11.8.

Accedemos a la liga <https://developer.nvidia.com/cuda-11-8-0-download-archive>, y elegiremos las características de nuestra instalación para poder bajar el archivo de instalación.

The screenshot shows the NVIDIA CUDA download page. It has a form with the following sections:

- Operating System:** Linux, Windows (selected)
- Architecture:** x86\_64 (selected)
- Version:** 10, 11, Server 2016, Server 2019, Server 2022 (selected)
- Installer Type:** exe (local), exe (network) (selected)

Below the form, there is a green bar that says "Download Installer for Windows 10 x86\_64". Underneath, it says "The base installer is available for download below." There is a section for "Base Installer" with a "Download (3.0 GB)" button. Below that, it says "Installation Instructions:" followed by two steps: "1. Double click cuda\_11.8.0\_522.06\_windows.exe" and "2. Follow on-screen prompts".

Figura 61. Pantalla de instalación de NVIDIA.

La instalación es muy amigable, si se quiere comprobar que se instaló abrimos una ventana de prompt, escribimos: `nvcc -V` y el comando debe de responder como en la figura.

```
Anaconda Powershell Prompt
(base) PS C:\Users\LANCIC-2022> nvcc -V
nvcc: NVIDIA (R) Cuda compiler driver
Copyright (c) 2005-2022 NVIDIA Corporation
Built on Wed_Sep_21_10:41:10_Pacific_Daylight_Time_2022
Cuda compilation tools, release 11.8, V11.8.89
Build cuda_11.8.r11.8/compiler.31833905_0
(base) PS C:\Users\LANCIC-2022>
```

Figura 62. Pantalla de verificación de la instalación de Cuda NVIDIA en el equipo.

#### A.1.2.2. Python.

Para facilitar el manejo de librerías y requerimientos se instaló Anaconda 3. No se requiere ningún prerequisite para esta instalación y, como no es complicado no se guiará en esta tesis su instalación. Si en nuestro ambiente no solo vamos a desarrollar esta solución, sería conveniente que tengamos varios ambientes separados, no vamos a decir cómo hacerlo, solo diremos que es posible habilitar varios ambientes para no mezclar librerías de diferentes proyectos.

Al final solo hay que verificar que tenemos acceso a Python, la forma más sencilla será ir al menú de inicio y buscar el programa instalado:

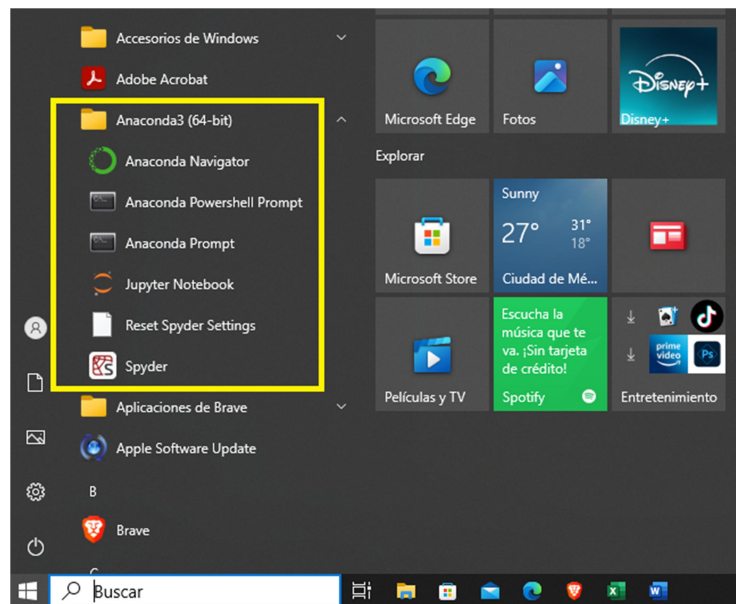


Figura 63. Pantalla de Windows para ubicar el buscador de software.

Pulsamos en Anaconda Powershell Prompt y en la ventana que se ejecute escribimos:

> python --version

```
Anaconda Powershell Prompt
(base) PS C:\Users\LANCIC-2022> python --version
Python 3.11.5
(base) PS C:\Users\LANCIC-2022>
```

Figura 64. Pantalla para corroborar que Python está instalado en el equipo.

Con esto habremos instalado correctamente python.

#### A.1.2.3. Pytorch.

Existen varias librerías para el desarrollo de redes neuronales en python, pero elegimos Pytorch por su facilidad y versatilidad, pues un modelo desarrollado en pytorch puede ser exportado a través

de Open Neural Network eXchange (ONNX) a TensorFlow o cualquier librería que lo pueda leer incluso en otro lenguaje.

La instalación de Pytorch puede o no usar Cuda, en nuestra instalación si lo usa. Lo primero es ir a la página de Pytorch <https://pytorch.org/> y buscar la siguiente tabla:

**NOTE:** Latest PyTorch requires Python 3.8 or later.

|                   |                                                                                     |           |                   |        |
|-------------------|-------------------------------------------------------------------------------------|-----------|-------------------|--------|
| PyTorch Build     | Stable (2.3.0)                                                                      |           | Preview (Nightly) |        |
| Your OS           | Linux                                                                               | Mac       | Windows           |        |
| Package           | Conda                                                                               | Pip       | LibTorch          | Source |
| Language          | Python                                                                              |           | C++ / Java        |        |
| Compute Platform  | CUDA 11.8                                                                           | CUDA 12.1 | ROCm 6.0          | CPU    |
| Run this Command: | conda install pytorch torchvision torchaudio pytorch-cuda=11.8 -c pytorch -c nvidia |           |                   |        |

Previous versions of PyTorch >

Figura 65. Pantalla de instalación de Pytorch.

En ella indicamos el tipo de instalación que queremos y nos da una línea de comando para poderlo instalar, la copiamos y en una ventana de Prompt de Anaconda la pegamos y la ejecutamos. En nuestro caso nos da la siguiente instrucción:

```
conda install pytorch torchvision torchaudio pytorch-cuda=11.8 -c pytorch -c nvidia
```

La ejecutamos y nos instalara todas las dependencias necesarias. Si queremos comprobar que se ha instalado Pytorch podemos crear un pequeño programa o verificar que está en la lista de librerías instaladas con el comando: `pip list | findstr torch`.

```
Anaconda Powershell Prompt
(base) PS C:\Users\LANCIC-2022> pip list | findstr torch
torch                2.1.0+cu118
torchaudio            2.1.0+cu118
torchvision           0.16.0+cu118
(base) PS C:\Users\LANCIC-2022>
```

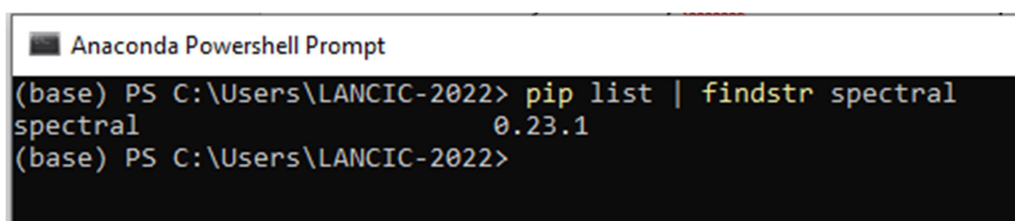
Figura 66. Pantalla de comprobación de librerías Pytorch instaladas en el equipo.

#### A.1.2.4. Spectral Python.

Vamos a usar una librería para poder leer imágenes Hiperespectrales que se llama Spectral Python.

Usaremos el mismo prompt de anaconda y escribiremos: `pip install spectral`.

La verificación la podemos hacer con el mismo comando



```
Anaconda Powershell Prompt
(base) PS C:\Users\LANCIC-2022> pip list | findstr spectral
spectral                0.23.1
(base) PS C:\Users\LANCIC-2022>
```

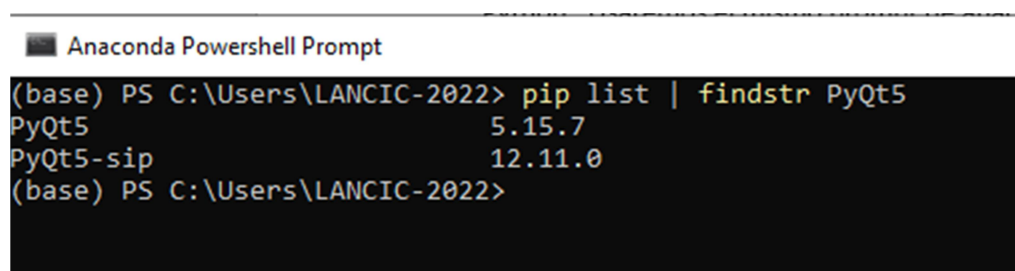
Figura 67. Pantalla con instrucción para instalar la librería Spectral Python.

#### A.1.2.5. PyQt5.

Hay varias librerías para desarrollar aplicaciones en ambientes de ventanas, uno de estos es PyQt5.

Con el podremos hacer una aplicación con una interfase de usuario amigable y fácil de usar. Por lo que la instalaremos en nuestro ambiente con el comando: `pip install PyQt5`

De la misma manera podemos buscar las librerías o hacer un pequeño programa.



```
Anaconda Powershell Prompt
(base) PS C:\Users\LANCIC-2022> pip list | findstr PyQt5
PyQt5                    5.15.7
PyQt5-sip                12.11.0
(base) PS C:\Users\LANCIC-2022>
```

Figura 68. Pantalla para instalar la librería PyQt5.

Estas son las dependencias básicas del programa que se desarrolló para este trabajo.

### A.1.3 Preparación de ambiente de desarrollo.

El programa de clasificación asume como existentes una serie de directorios donde deposita y busca diferentes archivos para procesar la estructura básica es la siguiente:

|                                                                                          |                        |                     |
|------------------------------------------------------------------------------------------|------------------------|---------------------|
|  HSCap  | 21/03/2024 12:36 p. m. | Carpeta de archivos |
|  data   | 30/01/2024 02:06 p. m. | Carpeta de archivos |
|  img    | 16/01/2024 12:27 p. m. | Carpeta de archivos |
|  logits | 13/03/2024 02:08 p. m. | Carpeta de archivos |
|  model  | 05/03/2024 10:02 a. m. | Carpeta de archivos |
|  utils  | 17/01/2024 11:29 a. m. | Carpeta de archivos |

Figura 68. Pantalla con la estructura de directorios necesarios para la ejecución del programa de clasificación.

El directorio donde se ponen los programas es HSCap. HSCap tiene 5 subdirectorios, el primero; data es donde se almacenan los histogramas de cada pigmento, el segundo; img es donde se almacenan las imágenes hiperespectrales (un archivo hdr y un su correspondiente dat) que serán procesados, el tercero; logits es donde se almacena la salida ideal de la clasificación de un pigmento, esta sirve para comparar en la clasificación. La cuarta; model es donde se almacenan los modelos entrenados para poder ser utilizados en la clasificación. En la quinta; útil se almacenan las funciones comunes en la implementación de los programas.

Finalmente, en HSCap hay 4 programas que son necesarios para la clasificación:

- **dataprocessing.py:** Es el programa que separa los pixeles de las áreas elegidas y los deja en un archivo único por pixel, para que pueda ser procesado. El programa discrimina los histogramas que no representativos del pigmento.
- **HSCap.py:** Es el programa que entrena la red neuronal y su fuente son los datos almacenados en el directorio data, una vez procesados y entrenado el modelo de la red neuronal, este es guardado en el directorio model junto con un archivo de metadatos que contienen el número de bandas, etiquetas de clasificación y los logit target para la comparación.

- **HSClas.py:** Es el programa que realiza la clasificación una vez generada el modelo ya entrenado, su entrada es la imagen hiperespectral que se desea clasificar y la salida es una imagen con las áreas etiquetadas de acuerdo con los pigmentos reconocidos.
- **HSmodel.py:** Es el programa donde se definen las arquitecturas de redes neuronales para la clasificación.

## Apéndice 2. K-medias (K-means)

K-means es un algoritmo que pertenece al área de “Aprendizaje No Supervisado”, ya que se aplica sobre conjuntos de datos que no están etiquetados. El objetivo es realizar  $n$  particiones de conjuntos de los datos suministrados, de acuerdo sus similitudes. En particular la implementación del algoritmo que utilizamos necesita 2 parámetros; un número  $K$  de particiones para agrupar y un número  $I$  de iteraciones para reagrupar los puntos del conjunto. A cada partición se le da el nombre de “cluster”, el cual es parametrizado por un vector  $\mathbf{m}_k$  llamado su media donde  $k$  está en el conjunto  $\{1, \dots, K\}$ .

Cada elemento del conjunto a ser agrupado es tratado como un vector cuya dimensión será identificada como  $\mathbf{s}$ , para este algoritmo se asume que los vectores están definidos sobre el campo de los reales y que se tomara la siguiente métrica entre dos puntos para establecer las cercanías:

$$d(x, y) = \frac{1}{2} \sum_{i=1}^s (x_i - y_i)^2 \quad (19)$$

Como indicamos al principio, lo primero es indicar el número K de “clusters” y el número I de iteraciones para agrupar sobre el conjunto objetivo, a continuación describiremos el algoritmo:

- Inicialización: Se establece un conjunto de K elementos, llamados centroides,  $\{m_1, \dots, m_k\}$  elegidos de manera arbitraria (puede ser una elección arbitraria o un método específico de muestreo de centroides).
- Asignación: Asigna a cada elemento a su centroide más cercano, y lo guarda con los elementos que son más cercanos a cada centroide.
- Actualización: En cada conjunto asignado se calcula un nuevo centroide y se reasignan los puntos al centroide más cercano.
- Iteración: **Se repiten** los pasos anteriores hasta que el movimiento de elementos no varíe o se llegue al número de iteraciones indicadas al principio.

Nosotros usaremos el algoritmo que viene en la librería “spectral pyton”, **ya que** lo que nos interesa es encontrar los “clusters” con mas elementos para el entrenamiento. El algoritmo es eficiente para este proposito.

Las ventajas del algoritmo que vamos a usar es que al darle un cubo de ancho x largo x bandas, nos regresa un rectangulo y en cada coordenada (ancho, largo) nos indica a que centroide pertenece.

## Bibliografía y referencias:

- [1] Awange Joseph, Paláncz Béla, Völgyesi Lajos(2025). Hybrid Imaging and Visualization. Springer Cham. DOI <https://doi.org/10.1007/978-3-031-72817-4>
- [2] Bernhard Schölkopf, Alexander J. Smola(2001). Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond. The MIT Press. DOI: <https://doi.org/10.7551/mitpress/4175.001.0001>
- [3] Charu C. Aggarwal(2020). Linear Algebra and Optimization for Machine Learning. Springer Cham. DOI: <https://doi.org/10.1007/978-3-030-40344-7>
- [4] Cristianini N, Shawe-Taylor J. An Introduction to Support Vector Machines and Other

Kernel-Based Learning Methods. Cambridge University Press; 2000. DOI:  
<https://doi.org/10.1017/CBO9780511801389>

- [5] Cybenko, G(1989). Approximation by superpositions of a sigmoidal function. Math. Control Signal Systems 2, 303–314. <https://doi.org/10.1007/BF02551274>
- [6] De Maesschalck R., Jouan-Rimbaud D., D.L. Massart,(2000). The Mahalanobis distance. Chemometrics and Intelligent Laboratory Systems,Volume 50, Issue 1.PP. 1-18. DOI: [https://doi.org/10.1016/S0169-7439\(99\)00047-7](https://doi.org/10.1016/S0169-7439(99)00047-7).
- [7] Esakkirajan S, Veerakumar T, Badri N Subudhi(2024). Digital Signal Processing. Springer Singapore. DOI: <https://doi.org/10.1007/978-981-99-6752-0>.
- [8] Fran Ramírez (2028). Historia de la IA: Frank Rosenblatt y el Mark I Perceptrón, el primer ordenador fabricado específicamente para crear redes neuronales en 1957 .<https://data-speaks.luca-d3.com/2018/07/historia-de-la-ia-frank-rosenblatt-y-el.html> (Consultado mayo 2024)
- [9] Hornik Kurt, Stinchcombe Maxwell, White Halbert (1989).Multilayer feedforward networks are universal approximators.Neural Networks,Volume 2, Issue 5, 1989, PP. 359-366,DOI: [https://doi.org/10.1016/0893-6080\(89\)90020-8](https://doi.org/10.1016/0893-6080(89)90020-8).
- [10] Kleynhans, T., Schmidt Patterson, C.M., Dooley, K.A. et al (2020). An alternative approach to mapping pigments in paintings with hyperspectral reflectance image cubes using artificial intelligence. Herit Sci 8, 84. <https://doi.org/10.1186/s40494-020-00427-7>
- [11] Krizhevsky Alex, Sutskever Ilya et al. (2012). ImageNet Classification with Deep Convolutional Neural Networks.Communications of the ACM, Volume 60, Issue 6. Pages 84 - 90. <https://doi.org/10.1145/3065386>
- [12] Kuo Jay C. -C.(2016). Understanding Convolutional Neural Networks with A Mathematical Model. arXiv:1609.04112. DOI: <https://doi.org/10.48550/arXiv.1609.04112>
- [13] Lecun, Y. (1989). Generalization and network design strategies. In R. Pfeifer, Z. Schreter, F. Fogelman, & L. Steels (Eds.), *Connectionism in perspective* Elsevier. <https://masters.donntu.ru/2012/fknt/umiarov/library/lecun.pdf>
- [14] Mahalanobis, P. C. (2018). ON THE GENERALIZED DISTANCE IN STATISTICS. *Sankhyā: The Indian Journal of Statistics, Series A (2008-)*, 80, S1–S7. <https://www.jstor.org/stable/48723335>
- [15] M.N. Murty, Rashmi Raghava(2016).Support Vector Machines and Perceptrons.Springer Cham.DOI: <https://doi.org/10.1007/978-3-319-41063-0>
- [16] McCulloch, W.S., Pitts, W(1943). A logical calculus of the ideas immanent in nervous activity. Bulletin of Mathematical Biophysics 5, 115–133. <https://doi.org/10.1007/BF02478259>

- [17] Neural Networks - History (2020). web site created for Eric Roberts' Sophomore College 2000 class. <https://cs.stanford.edu/people/eroberts/courses/soco/projects/neural-networks/History/history1.htm>
- [18] Ovidiu Calin(2020). Deep Learning Architectures - A Mathematical Approach. Springer Cham. DOI: <https://doi.org/10.1007/978-3-030-36721-3>
- [14] Python Tutorial - <https://www.w3schools.com/python/default.asp>
- [19] PYTORCH TUTORIALS - <https://pytorch.org/tutorials/>
- [20] Qt for Python Documentation - <https://doc.qt.io/qtforpython-5/contents.html>
- [21] Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6), 386–408. <https://doi.org/10.1037/h0042519>
- [22] Sanjeev Kulkarni, Gilbert Harman(2011).An Elementary Introduction to Statistical Learning Theory.John Wiley & Sons, Inc.DOI: 10.1002/9781118023471  
Teuvo Kohonen (2012). Self-Organizing Maps. Springer Berlin, Heidelberg. DOI: <https://doi.org/10.1007/978-3-642-56927-2>
- [23] Shannon C.E.(1948). A Mathematical Theory of Communication. AT & T Bell Laboratories.1948-10-01. pp. 623–656. [https://archive.org/details/sim\\_att-technical-journal\\_1948-10\\_27\\_4/page/622/mode/2up](https://archive.org/details/sim_att-technical-journal_1948-10_27_4/page/622/mode/2up)
- [24] Simonyan Karen, Zisserman Andrew(2014). Very Deep Convolutional Networks for Large-Scale Image Recognition Vol 12 Issue 08. SSRN Electronic Journal. 12. 301-307.
- [25] Szegedy Christian et al (2015). Going Deeper with Convolutions. 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). DOI: [10.1109/CVPR.2015.7298594](https://doi.org/10.1109/CVPR.2015.7298594)
- [26] Trevor Hastie, Robert Tibshirani, Jerome Friedman(2009). The Elements of Statistical Learning.Springer New York, NY.DOI: <https://doi.org/10.1007/978-0-387-84858-7>
- [27] Turing A.M (1950), Computing Machinery and Intelligence. Mind 49: 433-460.
- [28] Willman Joshua (2020). Modern PyQt - Create GUI Applications for Project Management, Computer Vision, and Data Analysis.DOI :<https://doi.org/10.1007/978-1-4842-6603-8>
- [29] Zendejas-Morales Claudia (2023) - Cursos de Quisquit - Cuaderno 5 día “Quantum Machine Learning”
- [30] Xian-Da Zhang(2021). A Matrix Algebra Approach to Artificial Intelligence. Springer Singapore. DOI: <https://doi.org/10.1007/978-981-15-2770-8>